МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Математическое моделирование
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Моделирование алгоритма компьютерного зрения для распознавания множества объектов на изображении и видеопотоке»

Студент В.Я. Олексюк
(И.О. Фамилия) (личная подпись)

Научный руководитель к.т.н., доцент, В.С. Климов
(ученая степень, звание, И.О. Фамилия)

Тольятти 2022

Содержание

Введение

Главной задачей компьютерного зрения является распознавание различных объектов на изображениях. На сегодняшний день до сих пор встречаются проблемы в локализации различных типов объектов. К таким проблемам относят неправильное расположение объекта под некоторым углом, излишки света в кадре или недостаток освещения, а также загромождение искомого объекта другим, но самой главной проблемой компьютерного зрения является производительность – время, затраченное на получение конкретного результата, в локализации искомого объекта на обученном алгоритме.

Актуальность настоящего исследования заключается в отсутствии готового решения для распараллеливания работы алгоритмов в распознавании множества объектов на изображении и видеопотоке.

Целью исследования является сравнение эффективности работы стандартного алгоритма Виолы-Джонса с многопоточной реализацией алгоритмов HOG с SVM путём подсчета скорости их работы.

Объектом исследования является сравнительный анализ алгоритма Виолы-Джонса и связки HOG с SVM.

Предметом исследования является оптимизация процесса локализации лиц на видеопотоке.

Гипотеза исследования состоит в том, что эффективность работы алгоритма Виолы-Джонса может быть увеличена, если будет разработана и реализована соответствующая модель по распараллеливанию процесса обработки изображения.

Для достижения поставленной цели необходимо решить следующие задачи:

– провести обзор существующих подходов по улучшению алгоритма Виолы-Джонса, рассмотрев созданные модификации, алгоритмы фильтрации и методы распараллеливания;
– спроектировать стандартный алгоритм Виолы-Джонса и многопоточную связку HOG и SVM с использованием технологии распараллеливания;
– реализовать и протестировать спроектированные алгоритмы Виолы-Джонса и HOG с SVM для выявления прироста производительности.

Научная новизна данного исследования заключается в использовании современной технологии распараллеливания «multiprocessing» со своей собственной логикой обработки входного кадра изображения для локализации объектов разного рода на кадре изображения и видеопотоке.

Практическая значимость исследования состоит в разработке модификации связки алгоритмов HOG и SVM для локализации объектов разного рода, которую можно применять для улучшения производительности.

Положения, выносимые на защиту:

– в ходе исследования была разработана новая модель распараллеливания работы алгоритма SVM по локализации объектов с применением технологии распараллеливания «multiprocessing»;
– модифицированный алгоритм SVM показывает лучшие результаты в производительности по сравнению с стандартной реализацией алгоритма Виолы-Джонса.

Магистерская диссертация состоит из введения, исследования предметной области, проектирования алгоритмов по локализации объектов, программной реализации и тестирования, а также заключения.

В первом разделе рассматривается предметная область, исследуется методы по оптимизации работы алгоритма Виолы-Джонса, приводятся методы фильтрация входного кадра изображения, а также логические обработки данных методов, рассматриваются существующие алгоритмы на

сегодняшний день, приводятся их плюсы и минусы, также исследуются технологии распараллеливания на разных архитектурах.

Во втором разделе проводится проектирование алгоритмов локализации объектов на изображении и видеопотоке, а именно проектируется стандартная реализация алгоритма Виолы-Джонса, а также производится проектирование модифицированного алгоритма SVM с применением технологии распараллеливания «multiprocessing» в связке с алгоритмом HOG, также производятся теоретические вычисления производительности модифицированного алгоритма SVM.

В третьем разделе производится программная реализация спроектированных алгоритмов Виолы-Джонса и связки HOG с SVM для вычисления производительности модификации и сравнительного анализа данных алгоритмов.

Полученная в ходе выполнения данной работы модификация алгоритма SVM позволяет увеличить скорость распознавания объектов на изображении и видеопотоке с использованием архитектуры CPU без ухудшения точности распознавания алгоритма.

Работа состоит из 3 разделов, заключения, содержит 41 рисунка, 23 формул, список использованной литературы включает в себя 30 источников. Основной текст работы изложен на 73 страницах.

1 Исследование предметной области

На сегодняшний день существует множество различных подходов к упрощению человеческой жизни. Но не все подходы обладают достаточной эффективностью. Даже сейчас насколько не были бы отлажены алгоритмы, они всё равно требуют внимания человека, чтобы удостовериться, точно ли машина делает всё правильно.

Поставлена задача – проанализировать эффективность применения алгоритма Виолы-Джонса в задачах распознавания нескольких объектов на изображении и видеопотоке с применением многопоточности.

На данный момент есть разные методы для локализации объектов на кадре изображения, можно привести в пример Eigenface в основе которого лежит преобразовании изображения в базисных компоненты, ISODATA задействующий кластерный подход при заданном количестве групп объектов, ERDAS Imagine использующий отклик целевого вектора по спектру и уменьшающий фоновую составляющую [15].

В качестве алгоритм для исследования берётся метод Виолы-Джонса. Данные алгоритм был создан для локализации лица человека, но его стали применять для локализации не только лиц. Главным достоинством данного метода является то, что он может быстро работать на слабой аппаратуре [13]. Благодаря этому в двухтысячных он показывал достойные результаты по локализации объектов.

1.1 Описание работы алгоритма Виолы-Джонса

Порог точность анализируемых данных растёт, как и числовой показатель скорости их обработки, а коэффициент ошибки стремиться к нулю. Метод Виолы-Джонса нашел своё применение в данной области. Разрабатываемое приложение для исследования будет использовать данный метод. Изначально было подчёркнуто, что алгоритм Виолы-Джонса отлично

справляется на аппаратуре двухтысячных годов, так как он включает в себя достаточно тривиальные вычисления, что приводит к быстрому получению результата [8].

Алгоритм находит объекты на кадре изображениях с высокой точностью и низким количеством ошибок [3].

Метод Виолы-Джонса отрабатывает по следующему алгоритму: имеется входное изображение, преобразуемое в виде двухмерного массива, элементы массива – это яркость пикселов, в диапазоне от нуля до двухсот пятидесяти пяти. Если входное кадр изображения представлен в чёрно-белых тонах, то понадобится лишь одна двухмерная матрица, если кадр цветной, то такое идёт преобразование в три двухмерных массива, которые представлены форматом RGB [13].

Базис алгоритма Виолы-Джонса заключается в конвертировании черно-белого кадра изображения в интегральное представление (рисунок 1):

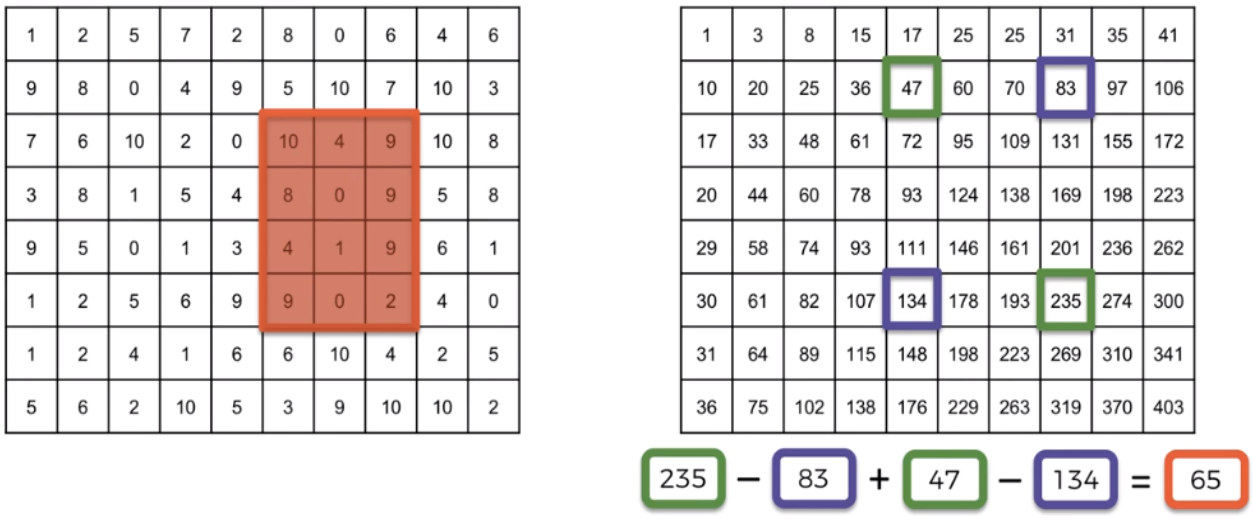

Рисунок 1 – Вычисления интегрального изображения с входного кадра

Это интегральное представление часто берут в использование другие методы по локализации, в пример можно привести преобразования-вейвлета [9]. В элементах интегрального представления содержится сумма

интенсивностей пикселов, расположенных с левой части и выше рассматриваемого элемента. Такая манипуляция даёт возможность найти сумму яркостей выбранного прямоугольника, в независимости от его размера, что очень подходит для применения примитивов Хаара. Размер матрицы интегрального представление совпадает с входным кадром изображения. Вычисление яркости в выбранном алгоритмом прямоугольнике находится по формуле (1):

$$II(x,y) = \sum_{i=0,j=0}^{i\leq x,\leq y} I(i,j), \quad (1)$$

где *I(i,j)* – это яркость пикселя входящего кадра изображения.

Элемент двухмерной матрицы II(x,y) является суммой пикселов в выбранной прямоугольной зоны берущий своё начало от элемента (0,0) до элемента с индексами (x,y). Время вычисления яркости в выделенной области линейно. Чтобы найти сумму яркостей пикселов, находящихся в зоне прямоугольника необходимо три арифметических вычисления и четыре обращения к массиву [6].

Возьмём прямоугольную зону ABCD (рисунок 2):

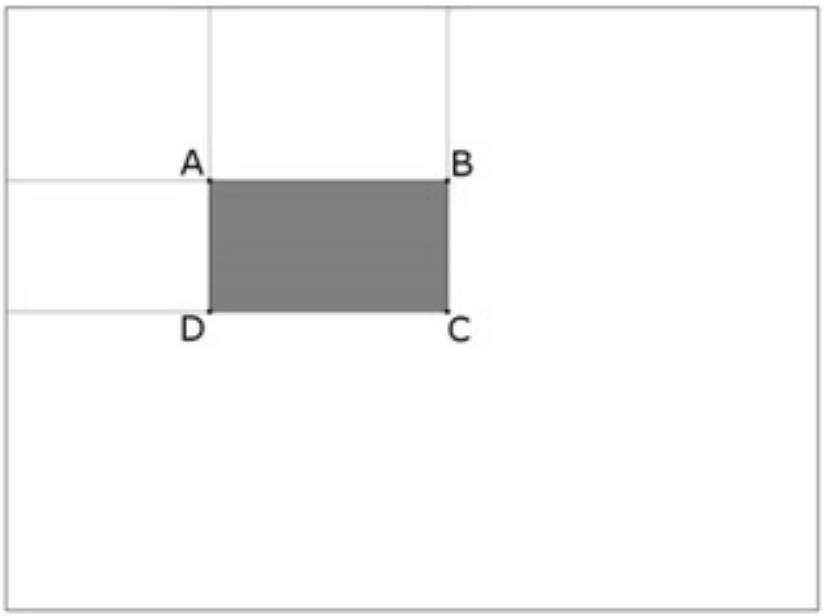

Рисунок 2 – Нахождение суммы яркостей пикселов в выбранной прямоугольной зоне

Находить значения яркости, находящихся в прямоугольнике пикселов, следует по формуле (2):

$$\sum ABCD = II(A) + II(C) - II(B) - II(D), \quad (2)$$

где *II(A), II(B), II(C), II(D)* – смежные прямоугольники расположенные рядом с выбранной зоной.

Алгоритм ищет некоторые характерных черт на кадре изображения. Характерные черты, применяемые методом Виолы-Джонса, могут немного отличаться от примитивов Хаара, но они могут иметь усложненной структуру и включать много соседних прямоугольников [9]. Интегральное представление пользуется примитивами Хаара. Они представляют собой бинарную аппроксимацию вейвлета Хаара. В обычном алгоритме Виолы–

Джонса применяются прямоугольные примитивы. Все признаки предполагает из себя бинарное черно-белое представление (рисунок 3):

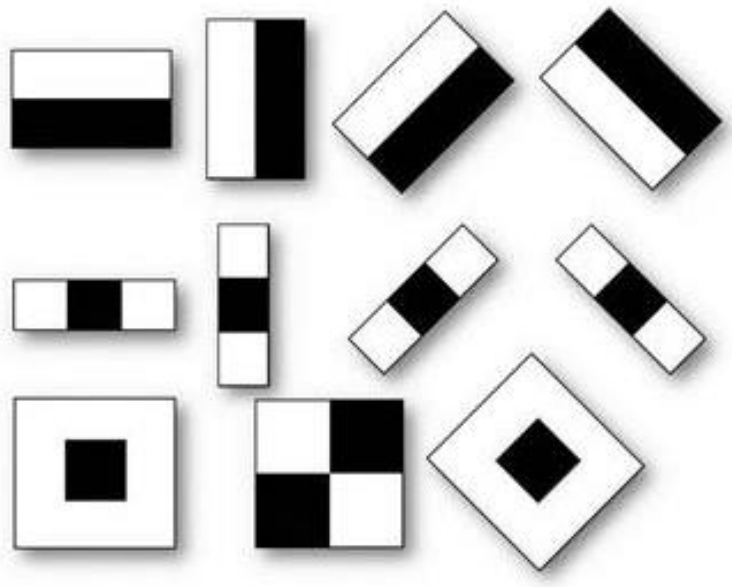

Рисунок 3 – Тривиальные примитивы Хаара

С помощью данных примитивов делается вывод - есть ли в выделенной области распознаваемый объект или эта область должна быть исключена из рассмотрения [18]. Вывод делается по вычисленному признаку по формуле (3):

$$F = X - Y, \tag{3}$$

где X – сумма пиксельных яркостей наложенной белой частью;

Y – сумма яркостей пикселей под черной зоной.

Значение функции сравнивается со значением в каскадном классификаторе, обученном находить определенный объект. Если значение больше определенного порога, то искомый объект находится в данной

области. Если значение меньше порогового, регион отбрасывается из рассмотрения [1].

Каскады классификаторов обучены с использованием алгоритма AdaBoost [18]. Это алгоритм машинного обучения, разработанный Йоавой Фройнд и Робертом Шапиром. Главное отличие от других алгоритмов машинного обучения заключается в том, что во время выполнения AdaBoost строит целую конструкцию базовых алгоритмов обучения для повышения производительности. Этот алгоритм достаточно адаптивен, так как последующий классификатор построен на объектах, которые предыдущие классификаторы не смогли правильно классифицировать. Любому выбранному для обучения объекту присваиваются числовые веса. Эти веса несут в себе особое значение для алгоритма. В момент, когда классификатор не идентифицировал объект – веса увеличиваются, заставляя классификатор учиться на своих ошибках.

Рассмотрим плюсы алгоритма AdaBoost:

- обобщение. Метод конструирует представления, дающие увеличение точности распознавания, превышающее качество основных методов, лежащих в его основе. Обобщение улучшается, дополнив алгоритм AdaBoost новыми лежащими в его основе алгоритмами.
- ресурсы, используемые алгоритмом AdaBoost, невелики. Время построения композиции зависит от времени обучения лежащих в ее основе алгоритмов;
- лёгкость реализации;
- может обрабатывать шумовые объекты.
- Недостатки алгоритма AdaBoost:
- восстанавливает каскадные классификаторы при значительных диапазонах шума в искомые данных. Функция экспоненциальных потерь, присваивающая значительно большие веса сложным объектам, где большинство базовых алгоритмов не работают. Часто

бывает, что трудоемкие объекты — это шум, и AdaBoost учится у них, что приводит к переучиванию. Решается эта проблема путем применения смягчающих функций;

– необходимо обеспечить AdaBoost большим количеством примеров;
– алгоритм сложения последовательностей может привести к неоптимальному набору базовых алгоритмов. Эта проблема решается путем восстановления к изначально сконструированным методам;
– построение громоздких композиций базовых обучающих алгоритмов.

Это приводит к увеличению времени обучения каскадного классификатора, а также к увеличению объема памяти для размещения сконструированных представлений. На рисунке 4 показан пример работы алгоритма Виола-Джонса с обученным каскадным классификатором на локализацию лиц:

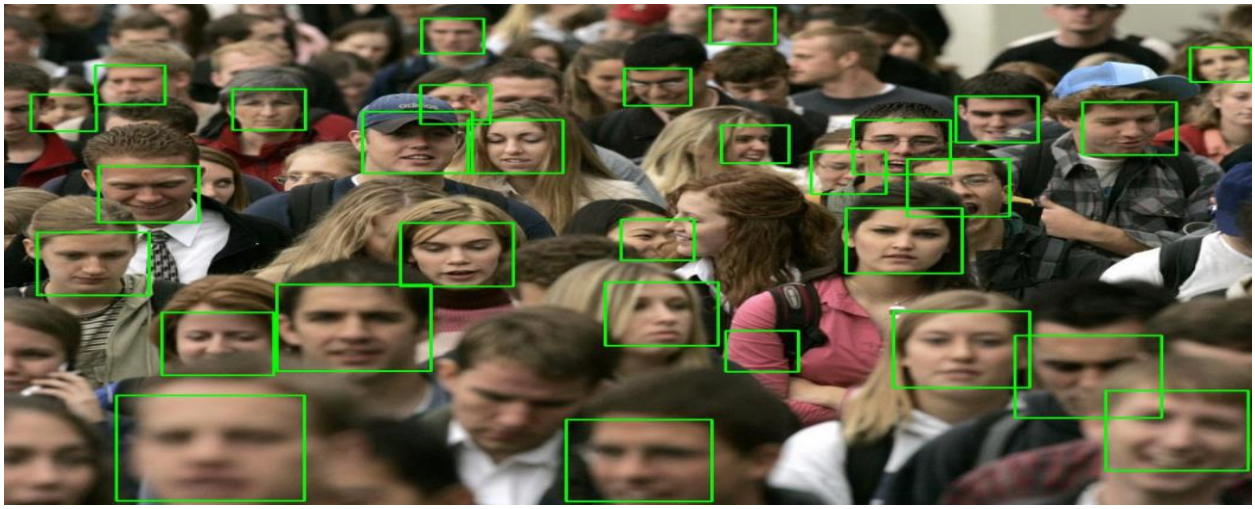

Рисунок 4 – Пример работы алгоритма Виолы-Джонса на распознавание нескольких лиц

На рисунке видно, что алгоритм не может локализовать всех людей, потому что они скрыты другими людьми. Алгоритм не плохо проявляет себя при размывании кадра изображения. Такая особенность заключается в хорошем обучен каскад классификаторов на значительном количестве выборки.

Основываясь на всех компонентах, можно выделить некоторые преимущества и недостатки алгоритма Виола-Джонса [2].

Преимущества:

– быстрая работа алгоритма;
– способность алгоритма локализовать любые объекты;
– низкое потребление вычислительных ресурсов;
– высокая точность локализации. Больше обучающая выборка, тем выше точность.

Недостатки:

– плохое реагирование на посторонние объекты в кадре, когда что-то заслоняет искомый объект;
– чувствителен к углу распознанного объекта;
– чувствительность к свету;
– необходимо много времени на обучения каскадных классификаторов.

Перечисленные минусы метода Виола-Джонса исправляются модификациями, а также включают в себя повышенную производительность.

1.2 Рассмотрение модификаций алгоритма Виолы-Джонса

Алгоритмы создаются для решения конкретных задач. У каждого из них есть свои особенности и недостатки, здесь и метод Виола-Джонса не остался в стороне. Но прогресс не стоит на месте, и есть обновления алгоритма Виола-Джонса.

Есть несколько путей модернизации:

– предварительная обработка входного изображения;
– изменение алгоритма Виолет-Джонса.

Препроцессирование входного изображения включает в себя методы Гауссова сглаживания фильтров, поиски границ на входном кадре с помощью оператора Previt, Sobel или Laplace и метод Canny [8].

Модификации алгоритма Виола-Джонса включают метод RASW.

Рассмотрим некоторые способы модификации метода Виола-Джонса, представленные выше.

Время работы метода Виола-Джонс завязано на разрешении искомого кадра. При увеличении размера кадра изображения увеличивается и количество вычислений. Для этого метода было изобретено - Runtime Adaptive Sliding Window (RASW), его также называют окном сканирования. Где размер окна определяется числовым значением. Шаговое сканирование тесно связано с точностью локализации объекта, а также определяет полосу пропускания. Шаг слишком большой, тогда пропускная полоса увеличится, но точность локализации будет снижена, так как объекты могут не попасть в распознанный и такие объекты могут быть пропущены [9].

Для быстрого пропуска областей, не содержащих распознаваемых объектов, и тщательной проработки областей, в которых они находились, нужно было найти шаблон. В результате это позволило бы увеличить скорость сканирования, не уменьшая коэффициент качества. Плюсом также было бы то, что ошибка классификатора была бы сведена к обнаружению нераспознаваемого объекта в другом месте кадра, где его не должно было быть.

В обычном алгоритме Виола-Джонса сдвиг установлен на небольшое значение, так как высокое значение приводит к низкому качества локализации.

Разработчики метода RASW провели исследование и обнаружили связь между искомым объектом и порядковым значением классификатора каскада, в котором эта область считается исключением. В зонах, включающие лишь фоновые выделения отбрасывается раньше, чем рядом с объектом. Таким образом пришли к заключению, что чем меньше расстояние до распознаваемого объект, тем ступень выхода больше и обратно.

Метод RASW представлен на рисунке 5:

Require: scaled input image X, classifier cascade S
Ensure: vector V of detected faces (bounding boxes)

1: **for** $y \leftarrow 0$ to $X.height - subwindow.height$ **do**
2: $\quad \Delta_x \leftarrow 1$
3: $\quad$ **for** $x \leftarrow 0$ to $X.width - subwindow.width$ **do**
4: $\quad\quad \Delta_x \leftarrow \Delta_x - 1$
5: $\quad\quad \Delta_y[x] \leftarrow \Delta_y[x] - 1$
6: $\quad\quad$ **if** $\Delta_x = 0$ AND $\Delta_y[x] = 0$ **then**
7: $\quad\quad\quad$ *exit-stage* $\leftarrow S(x, y)$
8: $\quad\quad\quad$ **if** *exit-stage* $= n$ **then**
9: $\quad\quad\quad\quad R \leftarrow (x, y, width, height)$
10: $\quad\quad\quad\quad$ push R into V
11: $\quad\quad\quad$ **end if**
12: $\quad\quad\quad$ **if** *exit-stage* $< \Delta_{x,t1}$ **then**
13: $\quad\quad\quad\quad \Delta_x = \Delta_{x,max}$
14: $\quad\quad\quad$ **else if** $\Delta_{x,t1} \leq$ *exit-stage* $< \Delta_{x,t2}$ **then**
15: $\quad\quad\quad\quad \Delta_x = \Delta_{x,nom}$
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad \Delta_x = \Delta_{x,min}$
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **else if** $\Delta_x = 0$ **then**
20: $\quad\quad\quad \Delta_x \leftarrow \Delta_{x,min}$
21: $\quad\quad$ **else if** $\Delta_y[x] = 0$ **then**
22: $\quad\quad\quad \Delta_y[x] \leftarrow \Delta_{y,min}$
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: **end for**

Рисунок 5 – Алгоритм метода RASW

Смещение окна распознавания использует каскад классификаторов, который возвращает шаг выхода (строчка 7). Затем проверяется условие, что если шаг выхода равен n - количеству классификаторов в каскаде, то объект обнаружен (строчка 8) и местоположение нынешнего окна сохраняется в вектор V (строчка 10). Сдвиг области способен принимать различные значения согласно оси абсцисс и ординат — это зависит от шага выхода предыдущего окна. Для того чтобы сдвиг зоны распознавания была правильным, следует сохранить сведение об выходящем шаге предшествующей воззрению распознаваемой области. При каждой итерации метода сдвиг по оси абсциссы и оси ординат уменьшается на один, в момент равенства обеих переменных в нулевом значении, на зону распознавания будет наложен каскад

классификаторов [19]. В случае только одного значения равно нулю, то такое значение станет базовым для минимального значения шага (строчка 20 и 22) [8].

Преимуществом метода RASW является ускорение алгоритма Viola-Jones без потери качества распознавания - быстрое перемещение окна распознавания по областям, не содержащим объекта и медленное движение в его близости.

Недостатки метода:

– сложная реализация;

– влияние шумовых объектов на зону улучшения данного метода. Чем больше шумовых объектов, тем ниже скорость;

– последующей модификацией является популярный метод, разработанный Джоном Канни в 1986 году. Он называется "Метод извлечения границы Canny".

Алгоритм состоит из пяти шагов:

– сглаживание. Размытие изображения для устранения шумов;

– нахождение градиентов. Поиск наибольших величин градиентов на входном кадре и нанесение границ;

– удаление минимальных градиентов. Максимальные значения локальных градиентов помечаются как градиенты;

– отфильтровывание ребер. Поиск значимых границ, являющимися порогами;

– коррекция неоднозначности. Границы определяются путем подавления всех рёбер, не относящихся к определённой границе.

При создании этого метода положили в основу следующие критерии:

– робастность к шуму;

– высококачественное распознавание областей искомых объектов;

– область одного объекта обязана задействовать лишь один отклик метода.

Эти критерии определяют функцию стоимости ошибки, с помощью минимизации которой находят значения линейного оператора, генерирующего свертку изображения [4].

Особенность Канни, по отношению к другим методам, является расчет градиента сглаженного кадра и избавление от дополнительных вершин, расположенных рядом у края, что привело к определению краёв, без заломов и разрывов, вблизи локальных максимумов градиента [3].

Исключение от незначительных граней было за счет использования двух лиминальных значений. Когда некоторые границы, обрабатываемые в целом по всему фрагменту, никогда не удовлетворяет условию перехода этих двух порогов, то этот фрагмент исключается из рассматриваемых. Это условие даёт возможность многократно уменьшить количество разломов на конечных границах. Такое избавление от шумовых объектов дает четкие и стабильные результаты, но увеличивает вычислительную нагрузку, создаваемую алгоритмом, и приводит к потере пограничных деталей. В результате углы искомого объекта на кадре исчезают, а границы в точках соединения исчезают [8]. Метод Канни работает по следующему алгоритму: первым шагом удаляются шумы, то есть - сглаживание, которое можно записать в виде формулы Гаусса для двухмерного случая:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \quad (4)$$

где σ – это коэффициент, отвечающий за сглаживание (чем больше, тем сильнее сглаживается входной кадр изображения).

После процесса сглаживания происходит вычисление градиентов изображения. Оно вычисляются двунаправленно, в вертикально и горизонтально. Такая операция базируется на первой производной. Значения, полученные в результате, вычисляются по формуле (5):

$$E_{ij} = \sqrt{{g_v}_{ij}^2 + {g_h}_{ij}^2}, \quad (5)$$

где g_v – вертикальное преобразование;

g_h – горизонтальное преобразование.

Направление искомых градиента находится по формулам (6):

$$\theta_{ij} = tan^{-1}\frac{{g_v}_{ij}}{{g_h}_{ij}}, \quad (6)$$

$$E_{T\,ij} = \begin{cases} E_{ij}\text{, если } E_{ij} > T, \\ 0 \text{ иначе} \end{cases}.$$

где θ_{ij} – векторный угол направления;

T – шумовой порог удаления.

Переменная T заносится, когда границы распознаваемого объекта выделятся и буту без изменений, после сглаживания [22].

Далее идёт операция пересечения минимальных градиентов. Это происходит зануление градиентов по двум значениям T1 и T2, с законченными условиями. Изучаемый градиент занулять запрещается, пока значение в заданной точке наименьшее, чем порог T1 и наибольшее, чем порог T2 [9]. Эти манипуляции приводят к нахождению нужных контуров на кадре.

Далее представлен пример работы метода Канни (рисунок 6):

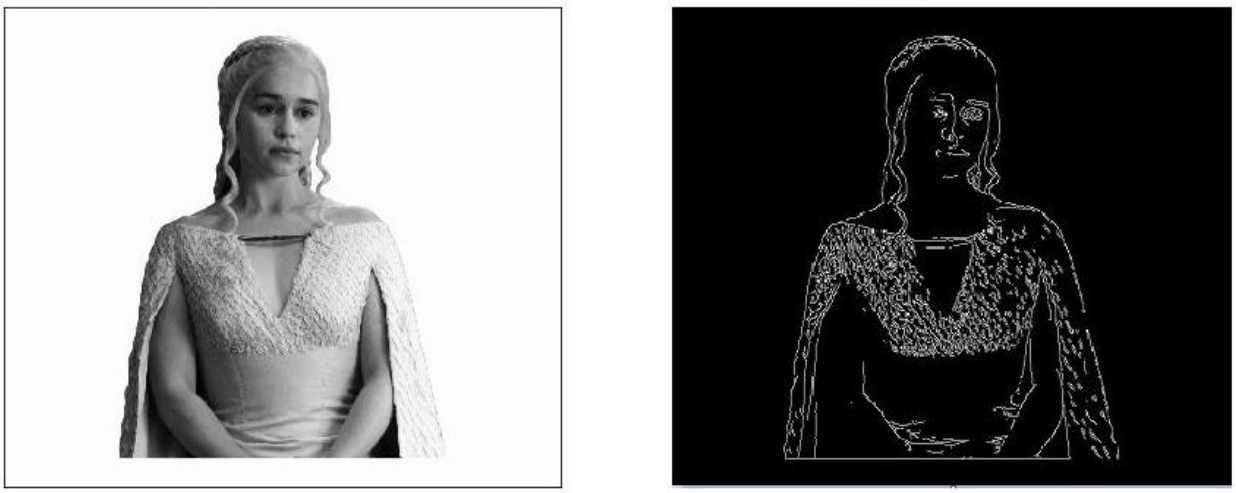

Рисунок 6 – Пример работы алгоритма Канни

Достоинством такого алгоритма Виолы-Джонса заключается в низком коэффициенте ошибки при нахождении объекта на кадре изображения [9].

Минусами являются:

– сложность вычислений;

– чувствительность к шумовым объектам.

В результате модификации выше демонстрируют существование способов улучшения алгоритма Виолы-Джонса и любого улучшение есть свои достоинства и недостатки, но в исследовании будет включена классическая реализация алгоритма Виолы-Джона.

1.3 Фильтрация входного кадра изображения

Существует разнообразные алгоритмы, которые позволяют выделить на кадре изображения интересующие области, без применения анализирующих методов. Большинство из этих методов применяет единое преобразование ко всем точкам на кадре изображения. На уровне фильтрации анализ входного изображения не производится, но точки, которые проходят фильтрацию,

можно рассматривать как области с особыми характеристиками, так как они являются, в каком-то роде, определением рассматриваемого объекта.

1.3.1 Бинаризация по порогу

Бинаризация изображения по порогу является самым тривиальным преобразованием. Для кадра изображения в RGB и градациях серого цвета порогом называют значение цвета [10]. Иногда бывают задачи, в которых достаточно лишь применить один раз бинаризацию по порогу, чтобы прийти к искомому результату. Примером такой задачи будет выделение объектов на белом листе бумаги (рисунок 7):

Рисунок 7 – Кадр изображения без применения бинаризации по порогу

Применяя бинаризацию по порогу, для выявления нужных объектов, получаем (рисунок 8):

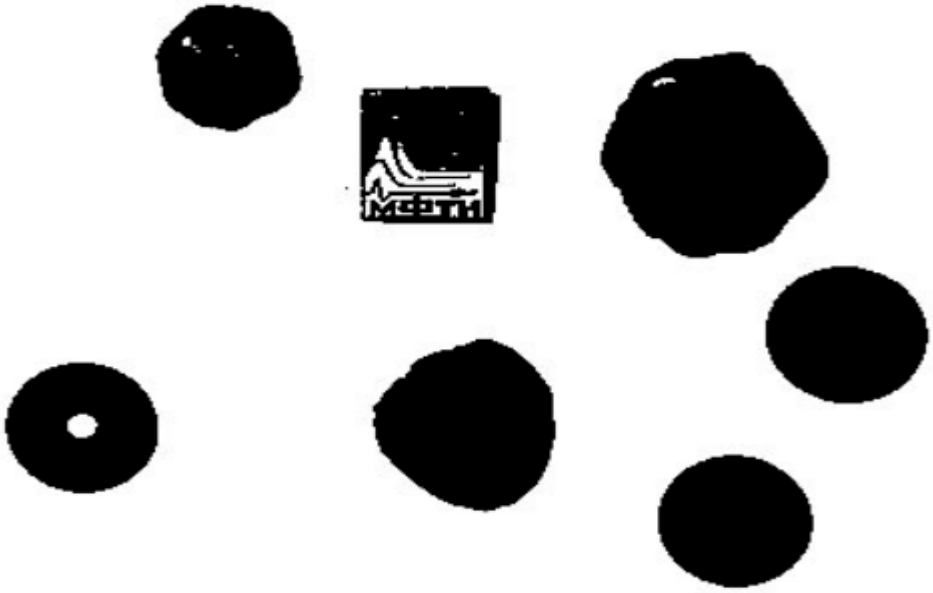

Рисунок 8 – Кадр изображения с применения бинаризации по порогу

Процесс бинаризации определяет выбор порога, по которому происходит бинаризация [14]. Искомый кадр входного изображения был преобразован по среднему цвету. В стандартном виде бинаризация осуществляется с помощью метода, который выбирает оптимальный порог. Таким методом может являться математическое ожидание, либо выбор наибольшего пика гистограммы (рисунок 9):

Рисунок 9 – Пик гистограммы для оптимального выбора порога бинаризации

Бинаризация входного кадра изображения может также применятся к изображениям не только в RGB, но и HSV формате. Это даст возможность

сегментировать необходимые цвета [14]. При таком использовании можно разработать детектор кожи человека.

1.3.2 Классическая фильтрация

Классические методы фильтрации Фурье, ФНЧ и ФВЧ, используемые при обработке сигналов также, можно использовать и при обработке кадров изображения для определения нужных элементов.

Популярным методом в работе с радиолокацией является метол Фурье, который используется при компрессии изображения — двухмерное преобразование Фурье по формуле (7):

$$G_{uw} = \frac{1}{NM}\sum_{n=1}^{N-1} \sum_{m=1}^{M-1} x_{mn} e^{2\Pi_j\left[\frac{mu}{M}+\frac{nw}{N}\right]}, \quad (7)$$

Пример работы метода Фурье (рисунок 10):

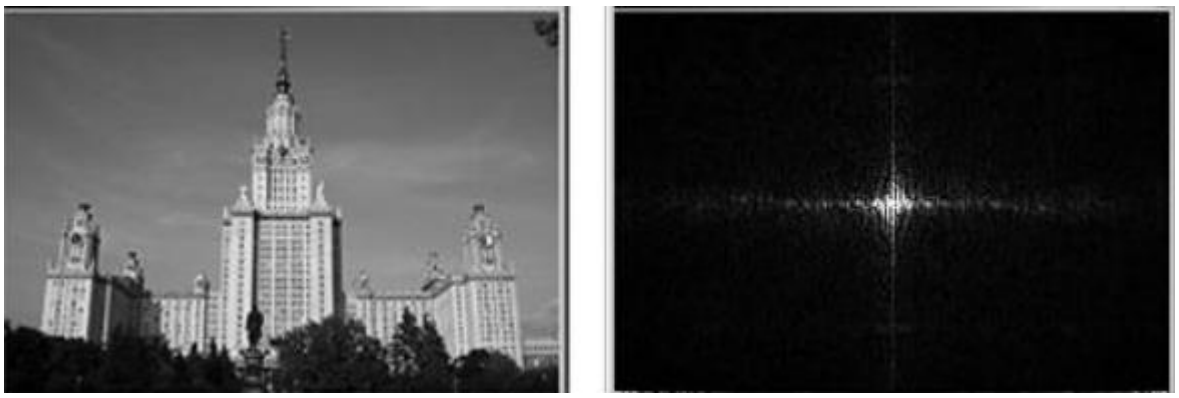

Рисунок 10 – Образ полутонового изображения с применением двухмерной фильтрации Фурье

Чтобы применить реализацию ФНЧ и ФВЧ, выбирается окно кадра изображения, далее идёт перемножение с фильтром Фурье схожего размера, таким образом получается свёртка. Свёртка характеризуется новым значением выбранной исходной точки [29]. При использовании ФНЧ и ФВЧ получаем следующие кадры от исходного изображения (рисунок 11):

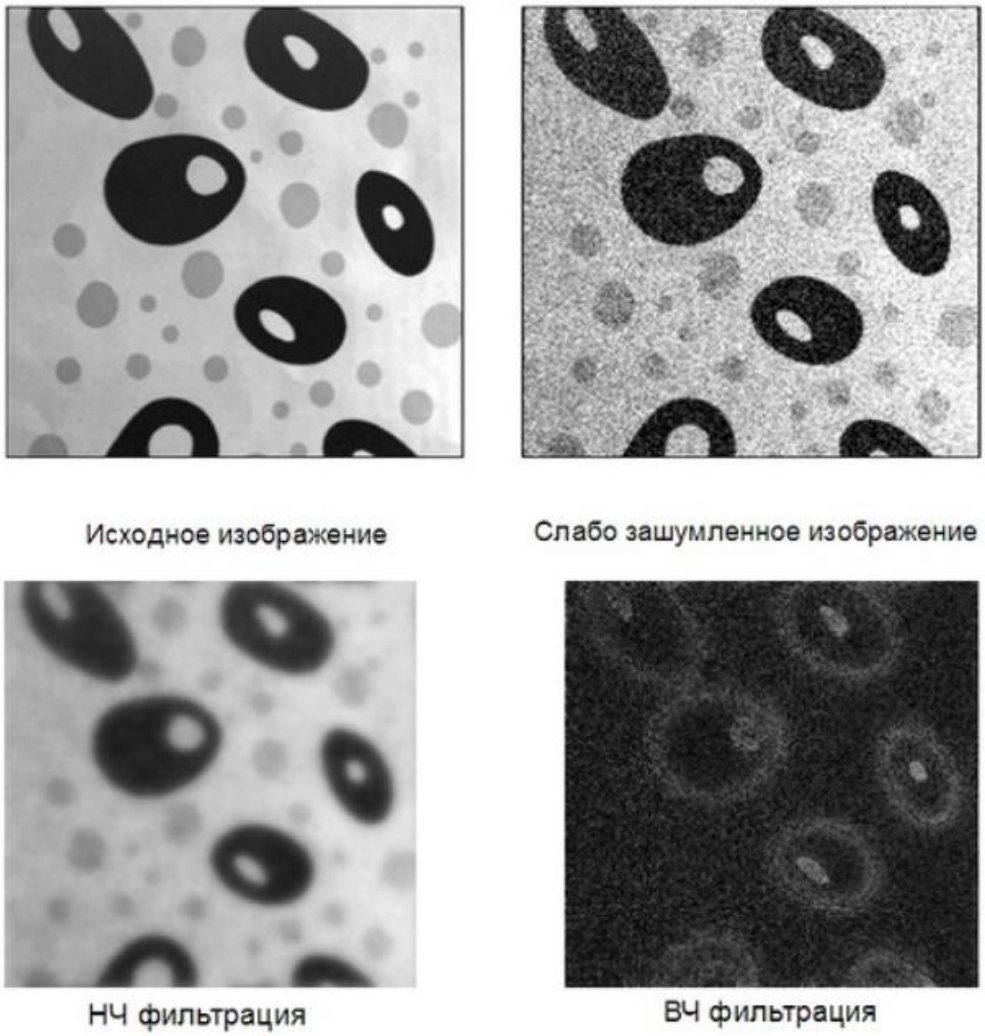

Рисунок 11 – Пошаговое применение ФНЧ и ФВЧ

Таким образом получается выявить рассматриваемые объектов лишь с применением фильтрации Фурье и использованием ФНЧ и ФВЧ.

1.3.3 Корреляция

Идея корреляционного метода заключается в измерении сдвига между наборами данных. В настоящее время корреляция является основополагающим фильтрацией для анализ входного кадра изображения, сжатие, измерение скорости объектов и измерение деформационных кадра изображения. Нелинейные оптимизационные подходы являются более простыми, но и время затратными и это иногда может является вопросом выбора более быстрых и более устойчивых линейных оптимизаций в фазовом

пространстве. Коэффициент взаимодействия корреляции r_{ij} определяется по формуле (8):

$$r_{ij}\left(u, v, \frac{du}{dx}, \frac{du}{dy}, \frac{dv}{dx}, \frac{dv}{dy}\right) = 1 - \frac{\Sigma_i \Sigma_j \left[F(x_i, y_j) - \bar{F}\right]\left[G(x_i^*, y_j^*) - \bar{G}\right]}{\sqrt{\Sigma_i \Sigma_j \left[F(x_i, y_j) - \bar{F}\right]^2 \Sigma_i \Sigma_j \left[G(x_i^*, y_j^*) - \bar{G}\right]^2}}, \quad (8)$$

где $F(x_i, y_j)$ – интенсивность пикселя в точке (x_i, y_j) на не деформированном изображении;

$G(x_i^*, y_j^*)$ – интенсивность пикселя в точке (x_i^*, y_j^*) на деформированном изображении;

$\bar{F}$ и $\bar{G}$ – среднее значение матриц интенсивности F и G.

Пример работы корреляционной фильтрации (рисунок 12):

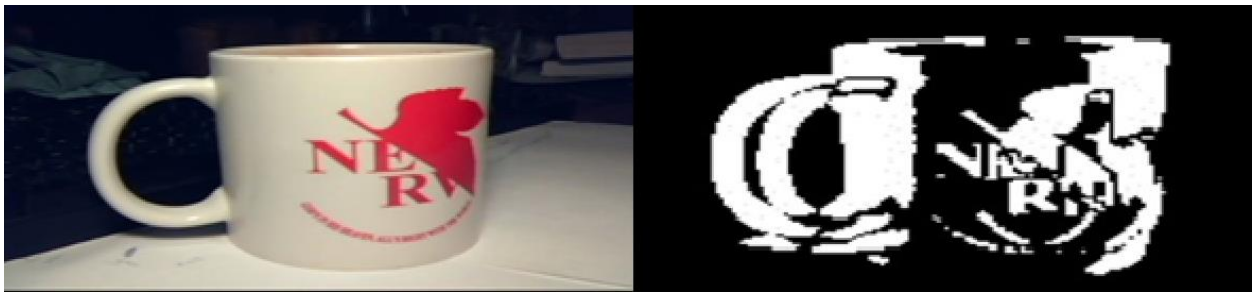

Рисунок 12 – Применение корреляционного метода к входному изображению

Также корреляционный метод лежит в основе вейвлетов Хаара, которые используются при в алгоритме Виолы-Джонса.

1.3.4 Фильтрации функций

Данная фильтрация представляет собой простые математические функции, которые позволяют обнаружить такие функции на изображении, как прямую, параболу, круг и так далее. Производится построение аккумулирующего изображение, в котором для каждой точки исходного кадра строится множество функций [21]. Стандартным методом при данной фильтрации является преобразование Хафа для прямых по формуле (9):

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right), \quad (9)$$

где r – длинна радиуса вектора, расположенного ближе к началу координат точки на прямой;

θ – это угол между осью абсцисс и данным вектором.

В данном фильтре для каждой точки (x_i, y_j) накладываются множество точек (a_i, b_j) прямой $y = a_i x + b_j$, только тогда, когда это равенство верно (рисунок 13):

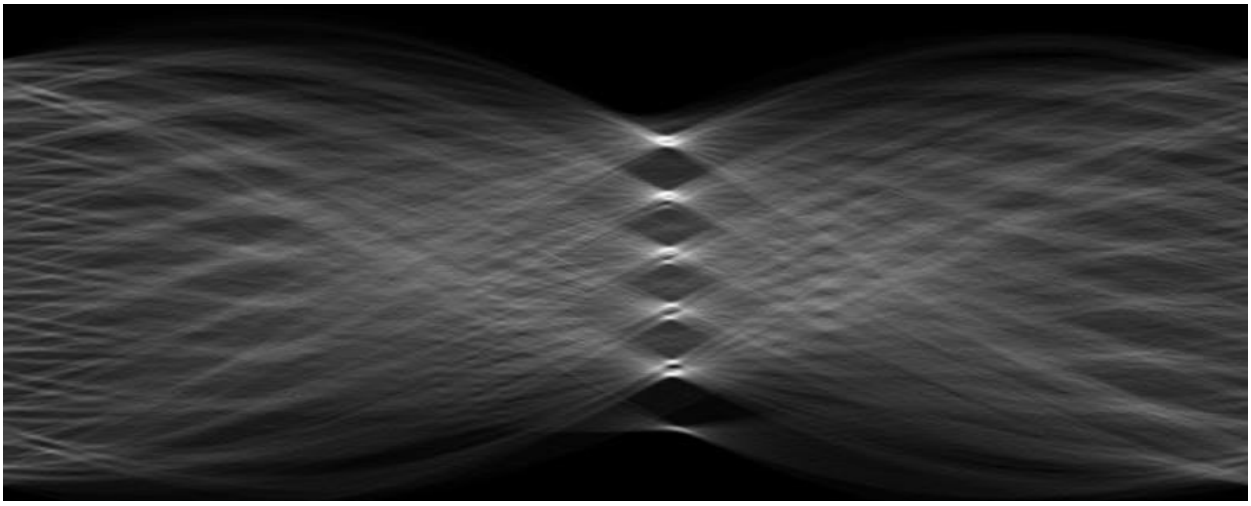

Рисунок 13 – Пример работы фильтрации функций

Также при данной фильтрации используют модифицированное преобразование, взамен преобразованию Хафа, которое позволяет искать любые фигуры на изображении, но такая замена несёт в себе значительные недостатки [18]:

– медленная скорость работы;

– высокая чувствительность к качеству бинаризационной обработки.

Преобразование Хафа заменяют иногда его аналогом – преобразование Радона по формуле (10):

$$R(s,a) = \int_{-\infty}^{t} (s\,cos(a) - z\,sin(a), s\,sin(a) + z\,cos(a))\,dz(1), \qquad (10)$$

Данный аналог вычисляется через быстрое преобразование Фурье, что даёт прирост к производительности при большом количестве точек, также его можно применять и к не бинаризированному кадру изображения, пришедшему на вход [17].

1.4 Логическая обработка результатов фильтрации

1.4.1 Морфология

Благодаря фильтрации мы получаем набор данных для анализа, но часто бывает, что данные нельзя использовать без постобработки и в этом помогают логические обработчики данных, применяемые после фильтрации.

Это простые операции наращивания и эрозии, которые способны убрать шумы из бинарного изображения, а также увеличить или уменьшить рассматриваемые элементы. Рассмотрим операцию наращивания (рисунок 14):

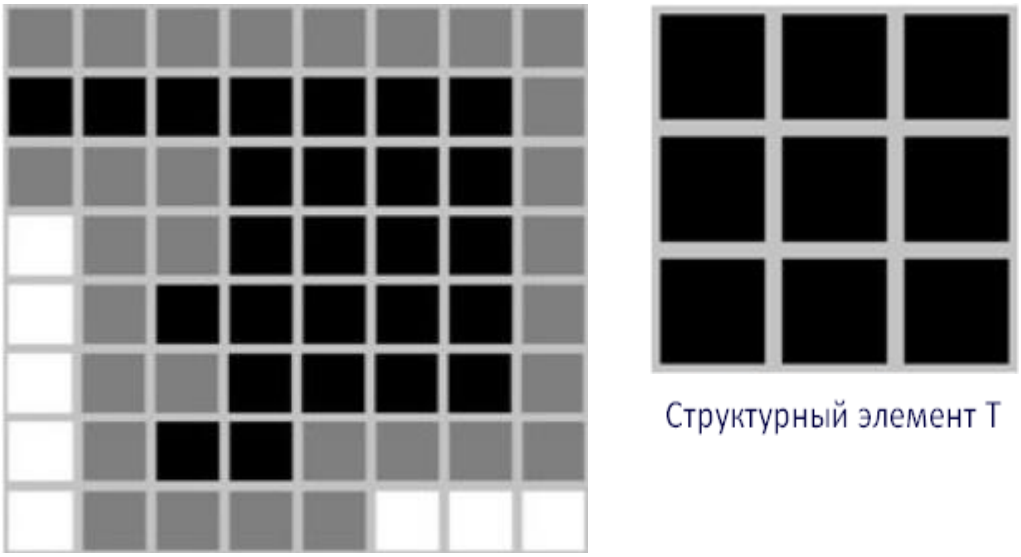

Рисунок 14 – Операция наращивания с использованием структурного элемента T

Структура T используется во всех пикселях бинарного изображения, если начало координат используемого элемента T совмещается с единичным бинарным пикселом, производится перенос и логическое сложение пикселов на бинарном изображении. В результате получаем запись логического значения в нулевые инициализации бинарного изображения [23].

При операции эрозии структура T пробегает все элементы бинароного изображения и выполняется логическое сложение центрального пиксела структурного элемента T с пикселом выходного кадра, только тогда, когда каждый единичный пиксел структуры T совпадает с единичным пикселом бинарного изображения (рисунок 15):

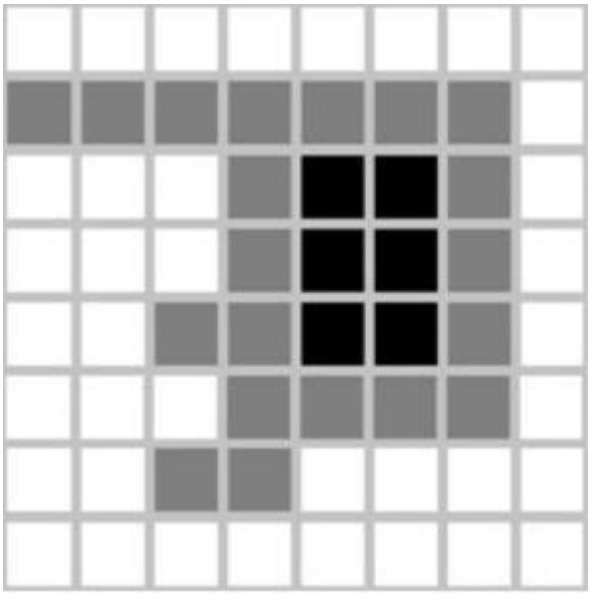

Рисунок 15 – Операция эрозия с использованием структурного элемента T

При использовании эрозии все объекты, которые будут меньше структурного элемента T, стираются, объекты, которые были соединены тонкими линиями распадаются и размер объектов уменьшается [8].

1.4.2 Контурный анализ

Контур характеризуется уникальностью объекта. Зачастую такая характеристика позволяет локализовать объект по контуру. Существует

обширный математический аппарат, который может позволить различать объекты между собой по контору, и он называется – контурный анализ.

Под контурным анализом базируется – цепной код Фримена. Данный код применяется для образования границ в отрезках – прямых линий определённой направления и длинны [30]. Код основан на четырех или восьми связной решетки. Длина каждого отрезка определяется решёткой, а направления выбреется кодом. Чтобы представления все направления в четырехсвязной решетке достаточно двух бит, а для восьми требуется три бита (рисунок 16):

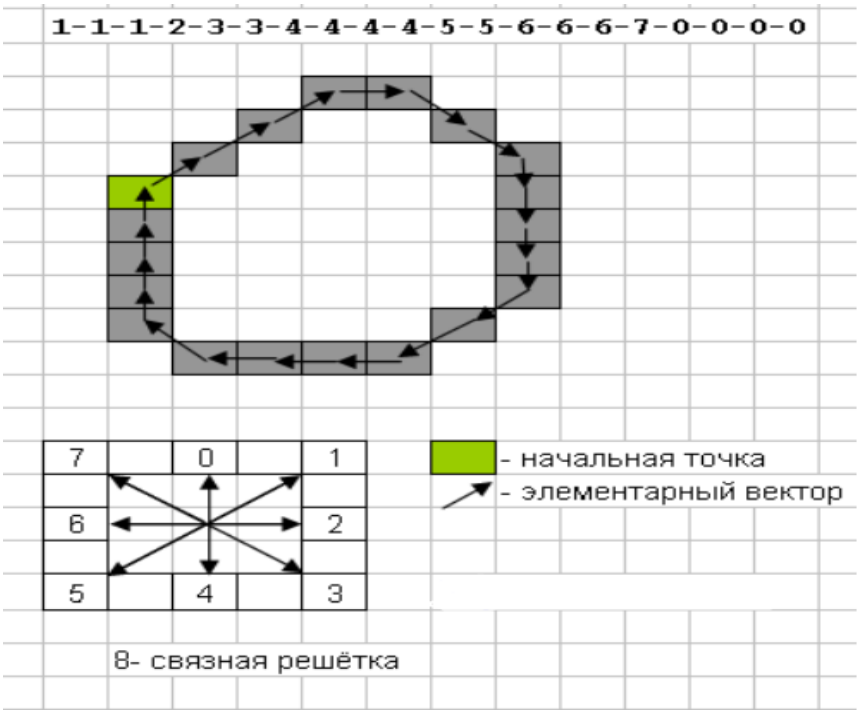

Рисунок 16 – Код Фримена

Данная логическая обработка имеет следующие недостатки:

– имеет сильную зависимость от шумов;

– может не найтись граница рассматриваемого объекта.

Контурный анализ является очень быстрым логическим обработчиком, благородя своему мощному и сбалансированному математическому аппарату [15].

1.5 Исследование существующих алгоритмов

1.5.1 FaceNet

FaceNet — это нейронная сеть, которая может преобразовывать входные кадры изображения лица в евклидово пространство, дистанция которого определяется мерой схожести разных лиц. Другими словами, лица ближе к друг другу, если они похожи [17].

Данная нейронная сеть использует функцию потерь, которую называют «TripletLoss». Эта функция сводит к минимуму дистанцию между якорем и кадрами изображений лиц, которые являются похожими друг на друга, и увеличивает дистанцию между разными изображениями. Формула «TripletLoss» выглядит следующим образом по формуле (11):

$$Loss = \sum_{i=1}^{N} \left[\left\| f_i^a - f_i^P \right\|_2^2 - \left\| f_i^a - f_i^n \right\|_2^2 + a \right]_+, \quad (11)$$

где *f(a)* – это энкодинг якоря;

f(p) – это энкодинг похожих лиц;

f(n) – это энкодинг непохожих лиц;

a – это константа, которая отбрасывает оптимизацию сети *f(a) — f(p) = f(a) — f(n) = 0*;

$[...]_+$ – это эквивалентно *max(0, sum)*.

Данная нейросеть также называется сиамской сетью. Сиамская сеть — это архитектуры нейросети, которая может дифференцировать входных данных [25]. В другом смысле позволяет отличать кадры изображения, которые являются похожими, от отличающихся кадров.

Сиамские сети включают в себя две идентичных нейронных сетей. Эти сети имеет одинаковые точные веса. Любая из сети принимает на вход одно из двух входных кадра изображений в качестве первичных данных. Затем результат последнего слоя, каждого изображения отправляются в функцию

«TripletLoss», которая определяет схожесть изображений. В FaceNet это производится путем вычисления расстояний между двумя выходами (рисунок 17):

Рисунок 17 – Пример работы FaceNet

Недостатки FaceNet:

– наличие светодиодных элементов на лице человека;
– нехарактерные для человека лица (надетая маска на лицо, нестандартные причёски, закрывающие лицо);
– погодные условия (влияние на уровень освещённости);
– угол размещения камеры.

1.5.2 HOG и SVM

HOG – метод основанный на гистограмме ориентированных градиентов. HOG дескрипторы используются следующими известными классификаторами: SVC, Random Forest, Boosting и так далее. В основе его лежит идея, о том, что распределение градиентной интенсивности позволяет точно определить наличие и форму объекта [20].

Метод HOG работает следующим образом: на вход поступает кадр изображения, этот кадр разбивается на несколько участков, которые называют ячейками. В этих ячейках идёт вычисление гистограммы градиентов внутренних точек. Такие гистограммы обычно компонуют в одну общую гистограмму $h = f(h_i, \dots, f_k)$, далее идёт нормализация по яркостям,

включающим в себя L_1 или L_2 нормы, и такая гистограмма принимает следующий вид по формуле (12):

$$h_{L_2} = \frac{h}{\sqrt{|h|_2^2+\varepsilon}}, h_{L_1} = \frac{h}{|h|_1+\varepsilon}, h_{\sqrt{L_1}} = \sqrt{h_{L_1}}, \quad (12)$$

где ε – константа малой величины.

Пример работы метода HOG с скользящим окном (рисунок 18):

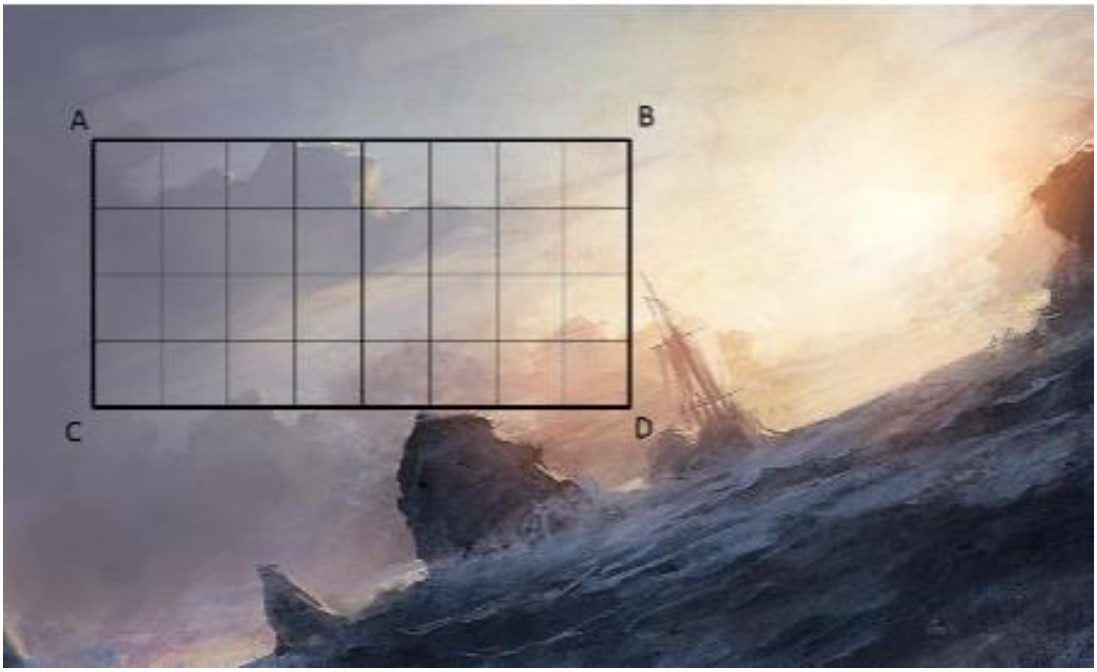

Рисунок 18 – Пример работы метода HOG

В результате, мы содержим данные пространственной информации о рассматриваемом фрагменте и инвариантны к освещению.

Далее идёт применение свёртки для вычисления градиентов с множеством ядер [-1,0,1] и транспонированных [-1,0,1], получается две матрицы D_x и D_y. Такие матрицы используются для вычисления величин и углов градиентов для каждой точки кадра изображения.

Допустим всевозможное множество углов разбивается на n равнозначных интервалов $\left(-\frac{k-1}{n}\pi, \frac{k}{n}\pi\right]$ и k включат в себя множество $\{1,\ldots,n\}$. Каждый интервал ставится в соответствие с бином гистограммы, следовательно гистограммная ячейка инициализируется величиной градиента и суммой с величиной бина, рассматриваемого интервала, включающего угол данного градиента [28].

По стандарту ячейки имеют прямоугольную форму, что позволяет использовать технику интегральных изображений.

Недостатки метода HOG:

– окклюзия – заграждение распознаваемого объекта другим объектом;

– чувствительность к шуму.

Метод HOG подготавливает вектор функции гистограммы направления градиента, по которому будет классифицироваться алгоритм SVM для обнаружения искомого объекта.

SVM – это машина опорных векторов. Данный алгоритм в основнои используется для классификации объектов, но также он подходит и для регрессии. Основная идея SVM заключается в нахождении гиперплоскости между входными данными, чтобы выделить объекты на изображении, на которых алгоритм был обучен [4].

Обычные алгоритмы обучается на распространенных характеристиках объектов, которые отвечают за различия одного класса от другого, но SVM работает совсем иначе. Данный алгоритм ищет наиболее схожие примеры между представленными классами – это и является опорными векторами. Другими словами, SVM изучает сходства между объектами, а другие алгоритмы ищут различия.

Алгоритм SVM в двухмерном пространстве основан на следующих шагах:

– выбор две оптимальные гиперплоскости, которые разделяют входные данные (рисунок 6.3 – линии красного цвета);

– увеличение данных линий на максимальную длину;
– построение средней линии на основе двух линий красного цвета – оптимальный градиент решения.

Данные шаги представлены на рисунке 19:

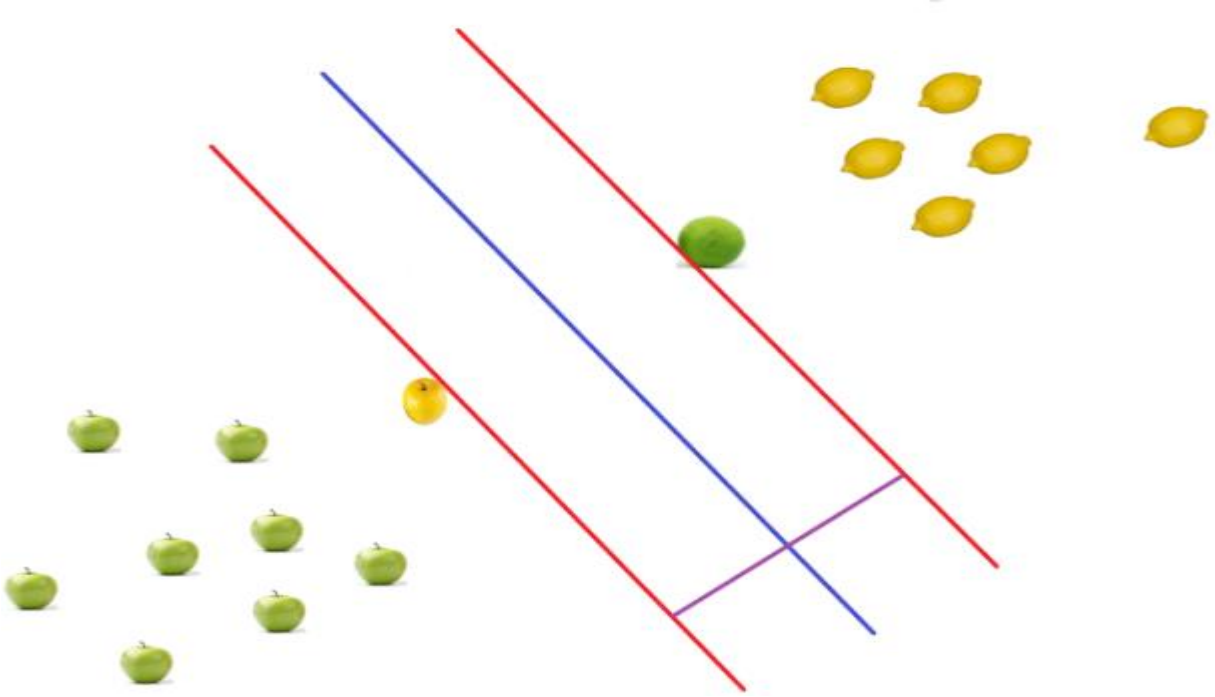

Рисунок 19 – Пример работы алгоритма SVM в двухмерном пространстве

На рисунке видно, что оптимальная гиперплоскость, отмеченная синим цветом, даёт четкое разделение между группой объектов разного класса, в данном случае яблока и апельсина.

SVM также поддерживает нахождение объектов нелинейного набора данных. В данном случае мы не можем найти прямую линию разделения объектов и на помощь приходит подход - трюк ядра. Основная мысль данного подхода заключается в добавлении дополнительного измерения к двухмерному пространству и получается трехмерное пространство, которое под собой несет уровни разделения объектов разный классов, другими словами, первый класс объекта будет на первом уровне, второй класс объекта

на втором. Пример нелинейного набора данных будет представлен на рисунке 20:

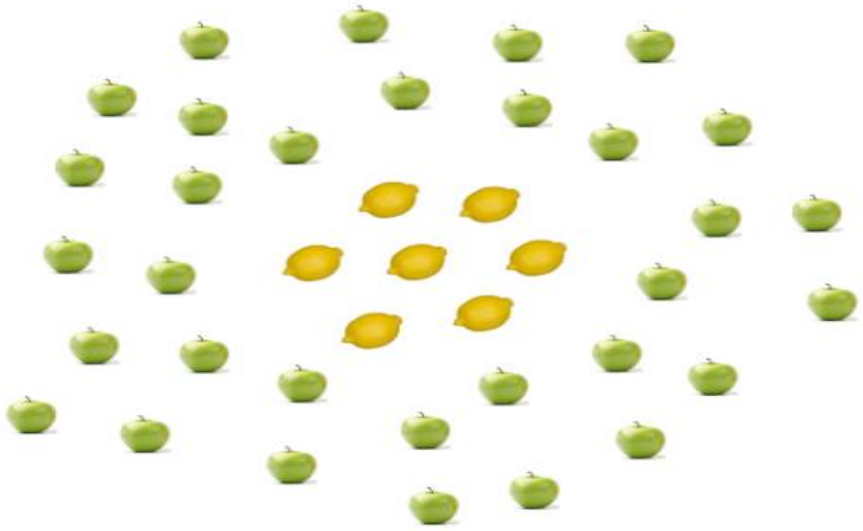

Рисунок 20 – Пример нелинейного набора данных для алгоритма SVM

Исходя из представленного рисунка, тут невозможно провести прямую линию, чтобы разделить два класса объектов и нужно прибегнуть к отображению двухмерного пространства в трехмерное рисунок 21:

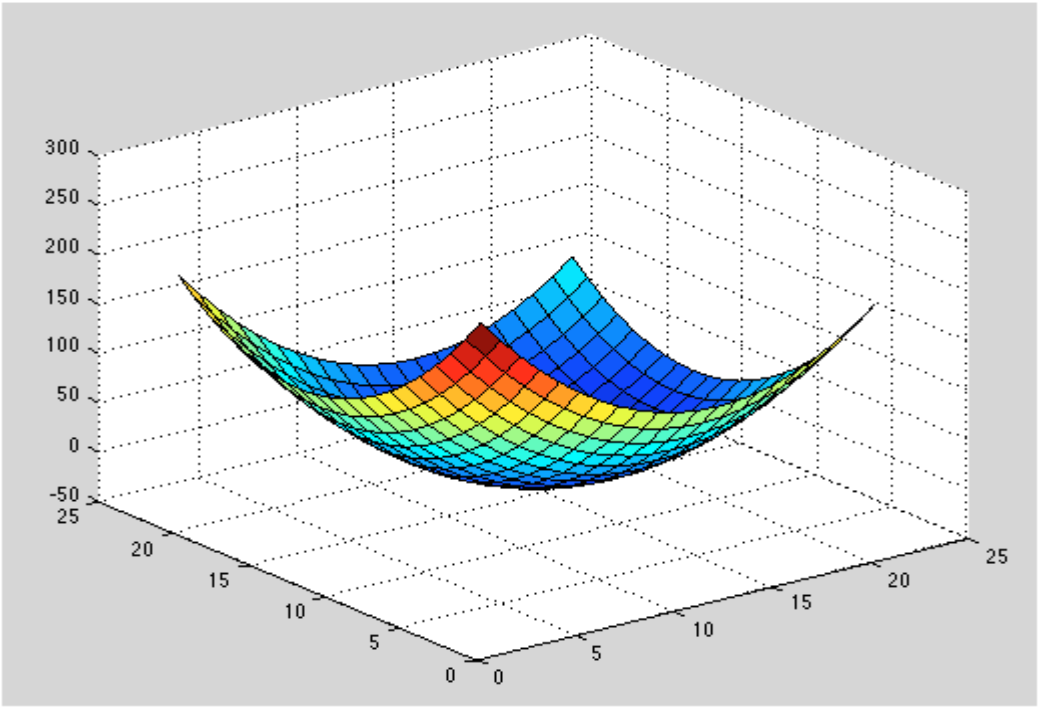

Рисунок 21 – Плоскость трехмерного пространства, одно из ядер алгоритма SVM

Данное трехмерное пространство строится на основе формулы (13):

$$z = x2 + y2, \quad (13)$$

В данном примере используется преобразования, в котором мы добавили уровни на основе расстояний осталось сопоставить наши классы с новым пространством. Если искомые объект находится в начале координат, то точка будет на самом низком уровне, когда мы удаляемся от центра, то мы поднимаемся по уровню выше. Исходя из вышеперечисленного получаем следующие результат (рисунок 22):

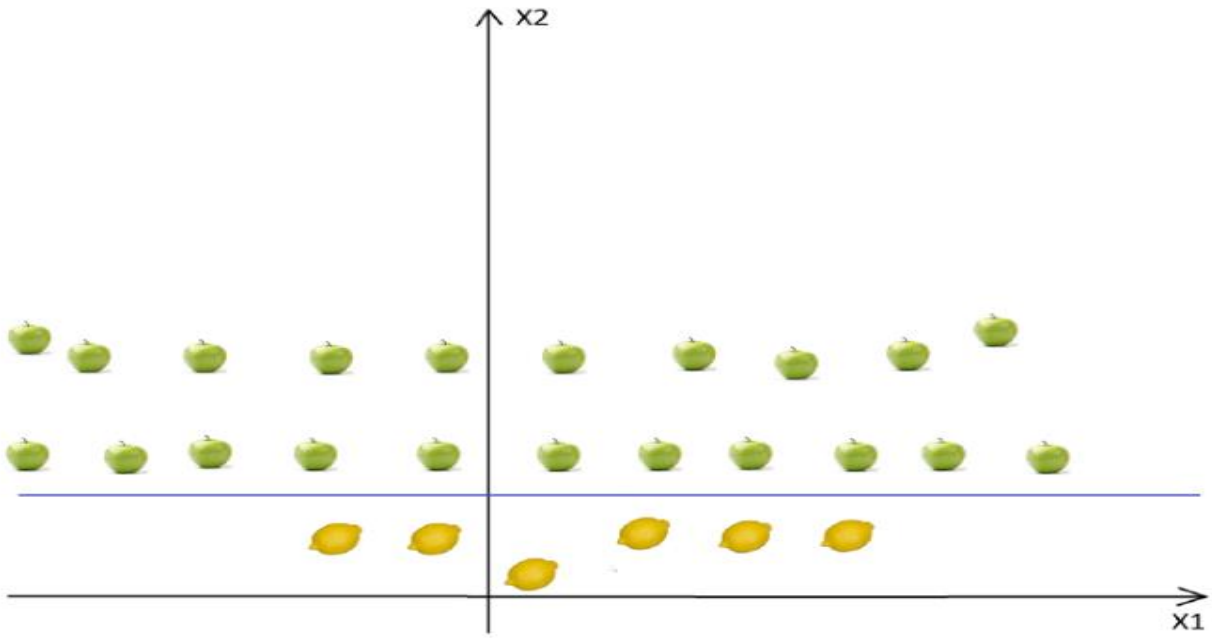

Рисунок 22 – Результат применения подхода SVM-ядра.

Недостатки SVM алгоритма:

– чувствительность к шуму;

– выбор правильного ядра является ресурсозатратным процессом;

– чем выше набор входных данных, тем дольше время работы.

1.5.3 MobileNet

MobileNet - это простая нейронная сеть, которая приспособлена для использования на мобильных, а также в приложениях глубокого обучения [16]. Стандартный процесс свертки основывается на ядре свертки m на m и одновременном просмотре соответствующих областей изображения. Свертка с разделением по глубине используя различные ядра свертки для преобразования различных входных каналов. Модель MobileNet включает в себя глубокую разделяемую свертку, которая может преобразовать стандартную свертку в глубокую свертку и точечную, где размер ядра свертки будет определятся, как один на один элемент (рисунок 23):

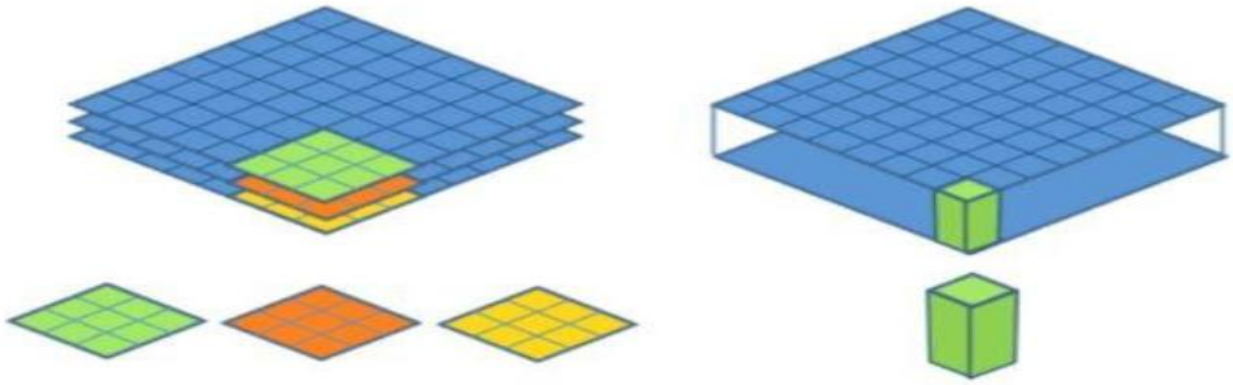

Рисунок 23 – Глубокая и точечная свертка модели MobileNet

В модели MobileNet базируется идея, что имеется вход:

$$N \times H \times W \times C, \tag{14}$$

где N – количество всевозможных выборок;

H – высота;

W – ширина;

C – количество каналов.

Глубина процесса характеризуется, тем, что вход $N \times H \times W \times C$ делится на C, а затем каждая параметр сворачивается размером три на три, что эквивалентно сбору пространственных характеристик каждого канала [15]. После чего применяется точечный процесс и выполняется k-раз. Конечный результат будет выглядеть следующим образом:

$$Depthwise + Pointwise - N \times H \times W \times k, \tag{15}$$

Модель MobileNets базируется на глубокой разделимой свертке, но нельзя забывать, что первый уровень является стандартной сверткой [10]. Существует также модифицированные версии модели MobileNet, всего их три

вида: MobileNetv1, MobileNetv2 и MobileNetv3. Каждая последующая версия является производительней предыдущей.

Недостатки:

– есть значительное потребление памяти при выводе результата (это исправлено последующими версиями);
– выполнение операции ReLU на малых размерностях может привести к потери информации, но на больших размерах, потеря будет не значительной.

1.5.4 FASTER R-CNN

Данная модель является улучшением Fast R-CNN модели, главным отличием является использование собственного метода локализации объектов взамен RPN системы. В основе RPN лежит система якорей (рисунок 24):

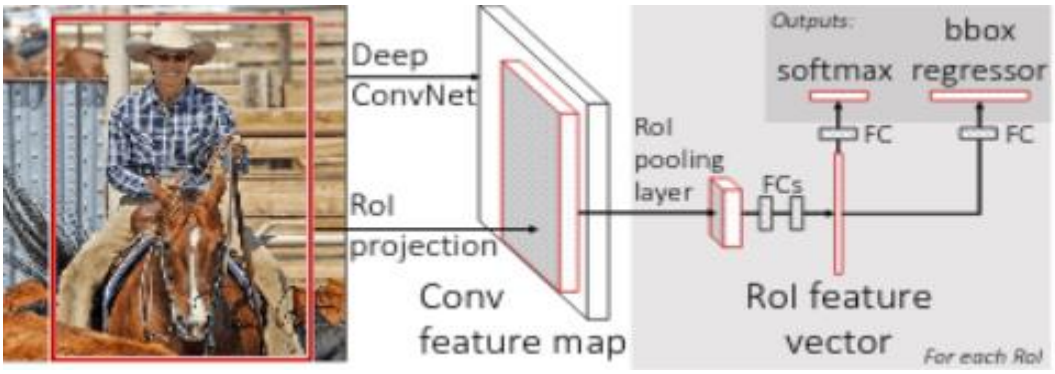

Рисунок 24 – Архитектура Fast R-CNN

Изображение попадает на вход сверхточной нейронной сети, после этого формируется карта признаков. Данная карта обрабатывается слоем RPN. Создаётся скользящее окно для входного кадра и запускается его прогон по карте признаков. Важно отметить, что центр этого скользящего окна связан с центром используемых якорей. Якорь – область разных соотношений сторон и разных размеров. В методе Faster R-CNN используется три соотношения и размерность стороны равной трём. Степень пересечения яркостей и истинных прямоугольников сравниваются с метрикой IoF, после чего выносится

решение о выбранном регионе, имеется ли там искомый объект или нет. После в дело вступает алгоритм Fast R-CNN. Берётся карта признаков с найденными объектами и передаётся в слой RoI, для дополнительной обработки полно связными слоями и классификацией, а также для определения смещения регионов потенциальных объектов.

Недостатки:

– faster R-CNN работает хуже с локализацией, чем предшественник Fast R-CNN;
– время обработки одного кадра не подходят для использования метода в реальном времени на видео потоке.

Fast R-CNN значительно подходит больше к обучающим и тестовым сессиях по R-CNN. Производительность Faster R-CNN во время тестирования, с включёнными регионами значительно замедляет алгоритм по сравнению с неиспользованием регионов. Поэтому регионы в алгоритме Faster R-CNN сильно влияют на его производительность [6].

1.6 Технологии разработки параллельного программного обеспечения

На сегодняшний день существуют разнообразные подходы по распараллеливанию вычислительных мощностей компьютера для увеличения скорости обработки информации с целью получить прирост производительности.

1.6.1 Технологии распараллеливания на CPU

Под CPU (Central Processing Unit) компьютера принято понимать под собой микросхему. Данная микросхема отвечает за выполнение операций над данными и управление устройствами, подключенных к персональному компьютеру. CPU представляет собой кремниевый корпус, называемом

кристаллом. На данный момент рынок компьютерных комплектующих занимают два гиганта Intel и AMD, которые участвуют в гонке за производительности.

CPU оснащён самый главный элементом – ядром. Ядро представляет из себя часть процессора, которая выполняет в себе один поток команд. Данные ядра могут быть весьма различны. Их различия заключаются в размере кэш памяти, частоте шины, технологии изготовления и во многом другом. С течением времени, благодаря развитию технологий, процессоры стали использовать в своём строении несколько ядер, что дало значительное увеличение производительности CPU и выполнение сразу несколько задач одновременно. Процессоры, имеющие несколько ядер, могут быстрее справляться с архивацией разного рода файлов, декодированием видео потока и многое другое [5].

С появлением многопроцессорной архитектуры CPU появились также решения по использованию этих дополнительных мощностей в программировании:

- openMP;
- thread;
- ray;
- dask;
- dispy;
- pandarallel;
- joblib;
- multiprocessing.

Открытый стандарт OpenMP разработан в 1997 г. как API интерфейс, для создание многопоточных программ базирующийся на языке Fortran, C и C++ [23].

В основе OpenMP лежит создание групп потоков, когда это необходимо. Работа OpenMP заключается в разветвлении между потоками выбранного

участка и выделение его границ, после окончания распараллеливания процесс можно опять вернуть к единому потоку выполнения программы. Данные группы потоков называют регионами. Данный подход прост в использовании и включает в себя два базовых типа конструкций групп:

– директивы «pragma»;
– функции среды OpenMP.

Директивы подсвечивают компилятору участок кода, где будут задействованы распараллеливающие регионы. Для вызова директивы необходимо использовать следующие ключевые: «#pragma omp».

Thread – класс, базовая конструкция множества языков программирования Java, C, C++, Python дающий возможность применять стандартные подходы к распараллеливанию частей кода. Данный подход имеет некоторый недостаток. Если распараллеливаемый кусок кода требует более сложный подход, то есть использование блокировок (синхронизация потоков), то придётся использовать дополнительные стандартные модификации языков программирования, но это не является проблемой, так как данные недостатки покрывают стандартные возможности любого языка [11].

Библиотека Ray создана для языка Python, группой исследователей Калифорнийского университета. Данная библиотека является основой для распределенных библиотек машинного обучения, но это не является её основной задачей. Абсолютно любую вычислительную часть кода на языке Python можно разветвить с помощью библиотеки Ray [1]. Синтаксическая составляющая данной библиотеки минимальна. Библиотека Ray включает в себя встроенный диспетчер кластеров для отладки и конфигурации, который может автоматически распределять выделенные узлы на вычисления при необходимости.

Библиотека Dask имеет сходство с Ray. Данная библиотека применяется для распределенных параллельных вычислений на языке Python и обладает

системой для планирования задач, с возможностью масштабирования до нескольких вычислительных машин.

Библиотека Dask работает двумя способами:

- распараллеливание структур данных, имеющая под собой собственную реализацию массивов данных NumPy, Lists или DataFrames;
- использование низкоуровневых механизмов распараллеливания, включающие декораторы функций, распределяющие задачи по узлам, которые могут возвращать результаты вычислений синхронным или асинхронным способом. Если необходимо использовать как синхронный, так и асинхронный подход, то библиотека Dask позволяет это сделать.

Ключевых различий между Dask и Ray, заключается в механизме планирования. Dask использует централизованный подход, который обрабатывает все вычисления кластеров. Библиотека Ray использует децентрализованный подход, на каждой вычислительной машине имеется свой планировщик [13].

Библиотека Dispy использовать полноценные программы на языке Python или конкретные функции для распределения по машинным кластерам. В его основе используется встроенные функции для сетевого взаимодействия между компьютерами с разными системами, и данная библиотека показывает себя очень хорошо при её использовании между такими ОС, как Linux, MacOS и Windows в одно время.

Библиотека Pandarallel занимается распределением вычислений на нескольких узлах для программной библиотеки Pandas – библиотека на языке Python для обработки и анализа данных. Pandarallel позволит распределить вычисления между ядрами процессора на компьютере.

В основе библиотеки Joblib лежит две основные идеи:

- выполнять задания параллельно;

– не делать пересчет результатов, если входные данные не поменялись.

Благодаря такму подходу Joblib подходит для использования в научных вычислениях.

Синтаксис Joblib является понятным для быстрого внедрения в своё решение, в основе его содержится декоратор, который используется для разделения вычислений между ядрами или для кэширования полученных результатов [27].

Joblib создаёт кеш для объектов Python, который в свою очередь помогает избегать лишний вычислений при дублированных входных данных, а также данный кеш помогает возобновить вычислений с места, где задание было приостановлено после ошибки программы. Кэш обладает оптимизацией для больших объёмов данных. Обрабатываемые данные могут быть доступными межу узлами процессов с помощью словосочетания «numpy.memmap».

Одним недостатком библиотеки Joblib является сложность реализации распределения вычислений между несколькими компьютерами.

1.6.2 Технологии распараллеливания на GPU

GPU (Graphics Processing Unit) – это микросхема, берущая на себя часть вычислительных процессов. Благодаря специализированной архитектуре, GPU подходит для вычислений с плавающей точкой, в свою очередь CPU подходит более для вычислений в многопоточном режиме [2].

GPU имеет возможность быстро проводить расчёты, где задействована одна и та же формула, к примеру нахождение точки затухания графики при проекции тени на текстурированный элемент сцены [7]. CPU подходит больше для проведения вычислений на нескольких ядрах одновременно. В большинстве случаев GPU используется для рендеринга графики. Также в его ответственности имеется вывод изображения на экран монитора, графического интерфейса ОС, а также любых других сторонних программ.

На GPU не накладывается строгого применения обязательства обрисовывать изображение на мониторе компьютера и рендеринга графики, существуют инструменты, способные применять мощности GPU для собственных вычислительных процессов, на своё усмотрение. Одними из таких инструментов имеет название CUDA и OpenCL [24].

CUDA – это библиотека прикладного программирования, позволяющая использовать вычислительные мощности GPU, созданные компанией «Nvidia» для языков программирования C, C++, Java, Python и т.д. Данный API-интерфейс предоставляет возможность разработчикам программных продуктов увеличивать производительность вычислительных мощностей используя GPU фирмы «Nvidia». Библиотека CUDA – даёт возможность доступа к виртуальным наборам команд GPU и их ядрам.

Доступность интерфейса Nvidia-API облегчает программистам взаимодействие с GPU, в отличие от такого API как «OpenGL», которые требут дополнительный опыт работы с графическим программированием. GPU на базе CUDA также дают возможность использовать такие программные среды, как «OpenACC» и «OpenCL» [24].

Компания Nvidia предоставляет также свой продукт под названием «CUDA Toolkit», который даёт возможность оптимизировать и разварачивать свои программные решения на встроенных системах с графическим ускорением, домашних компьютера, корпоративных терминалов по обработке массивных данных, на платформах облачных вычислений, а также суперкомпьютерах.

API-интерфейс CUDA обеспечивают быстрое интегрирование в различных областях, таких как линейная алгебра, обработки кадров изображения и видео потоков, глубокое машинное обучение и аналищ графов.

Программные решения на базе CUDA имеют возможность взаимодействия со всем семейством графических процессоров «Nvidia», доступных в домашних условиях и на терминалах в облачных средах, также имеется возможность масштабирования между несколькими GPU.

Далее будет представлена модель памяти CUDA (рисунок 25):

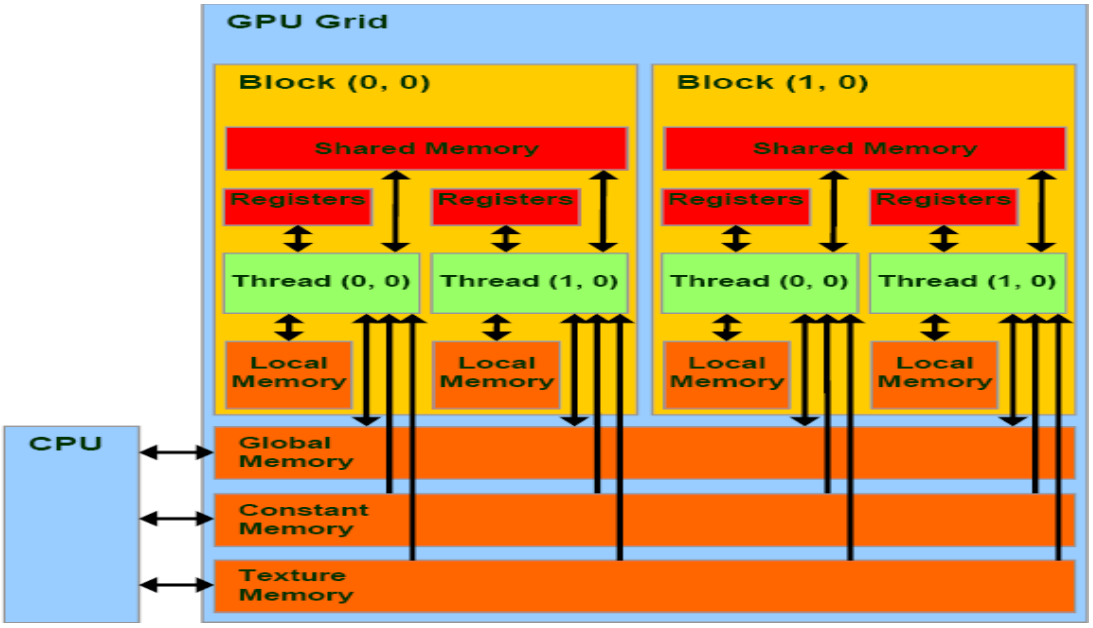

Рисунок 25 – Модель памяти CUDA

Необходимо знать, что CUDA работает с разными типами памяти. Локальная и глобальная память не кэшируется и исходя из аппаратных возможностей задержки при доступе к данной памяти выше, чем у регистровой памяти, потому что она в отдельных микросхеме.

Библиотека OpenCL – это API с открытым исходным кодом для вычисления на GPU. Данная библиотека не зависит от поставщика видеокарт и поддерживает многоядерные графические процессоры [14].

На сегодняшний день огромное количество GPU от компаний AMD и Nvidia имеют совместимость с OpenCL. Наличие установленного драйвера GPU и предустановленного SDK OpenCL являются достаточным условием для вычислений на GPU.

Библиотека OpenCl обладает рядом директив, позволяющих включать различные дополнительные функции для вычислений, их ещё называют наивными функциями.

Далее будет представлена модель памяти OpenCL (рисунок 26):

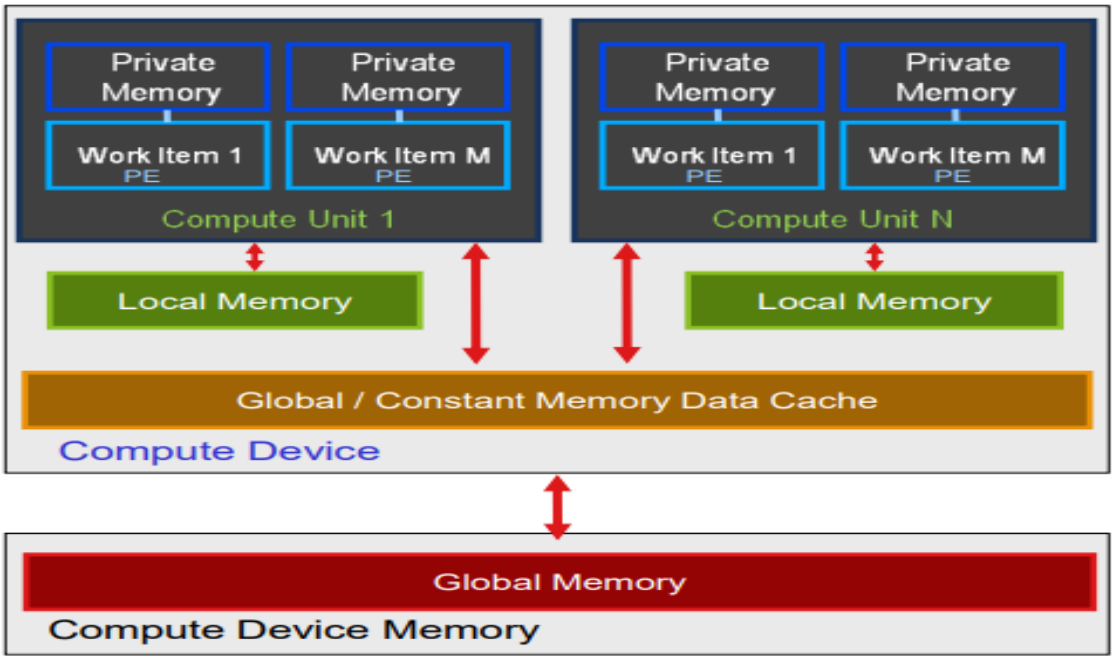

Рисунок 26 – модель памяти opencl

На рисунке представлена модель OpenCL с разделяемой памятью. Она также, как и CUDA имеет множество разных разделённых пространств. Приветное пространство предоставляется для одного рабочего процесса, локальное пространство имеет возможности, распространяемые на группы рабочих процессов, глобальное пространство доступно всем и последнее константное пространство, имеющее режим только чтения и также доступно всем [22].

2 Проектирование алгоритмов по локализации объектов

2.1 Проектирование однопоточной реализации алгоритма Виолы-Джонса

Для улучшения быстродействия алгоритма Виолы-Джонса будет задействована фильтрация входного кадра изображения с видео потока, а именно перевод изображение в градиент серого (рисунок 27).

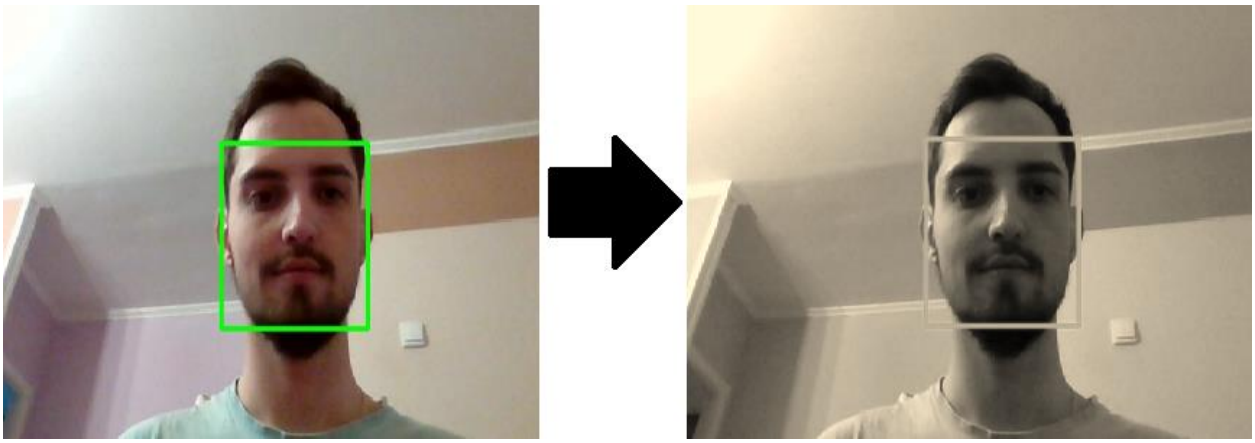

Рисунок 27 – Преобразование в градиент серого

Отдельный пиксель для градиента серого вычисляется по формуле (16):

$$Result = \frac{R + G + B}{3}, \tag{16}$$

где RGB – это аддитивная световая модель.

Для дополнительного эффекта быстродействия необходимо применить существующие технологии распараллеливания.

Предполагается, что программное решение с реализованным методом Виолы-Джонса будет иметь однопоточную реализации процесса распознавания, тогда как в свою очередь алгоритм SVM будет использовать

технологию многопоточного программирования и данная технология будет применяться к части, отвечающей за локализацию объекта. Созданный поток будет брать из пула памяти собственный каскад-классификаторов и применять его для обнаружения объекта на входном кадре изображения.

Чтобы реализовать алгоритм Виолы-Джонса необходимо построить блок-схему (рисунок 28):

Рисунок 28 – Блок-схема обобщённого варианта алгоритма Виолы-Джонса однопоточной реализации

На представленном рисунке блок, отвечающий за считывание количество кадров в обработанных программой за единицу времени в секундах, будет расположен до цикла, исходя из этого необходимо будет

выделить под данную операцию один поток, так как программа может работать с зависаниями, в однопоточном варианте. Данное введение даст возможность определения эффективности работы алгоритма Виолы-Джонса с применением локализации сразу нескольких объектов. Самые главные вычислительные процессы находятся в блоке работы алгоритма Виолы-Джонса – это место является подходящим для распараллеливания, так как алгоритм применяет прогонку объектов Хаара по изображению и этот процесс можно разделить между потоками. Исходя из этого можно сделать вывод о распределении между потоками собственного каскада-классификатора для локализации искомого объекта или группы объектов, либо распределение выполняемой работы на входном кадре изображения.

Применение технологий многопоточного программирования к блоку по локализации искомых объектов даст значительный прирост к производительности, в причину того, что один поток не сможет быстро обрабатывать входящие кадры изображения для нахождения заданных объектов через выбранные каскады классификаторов, что приведёт к ухудшению производительности, а именно уменьшению обработанных кадров за единицу времени в секунду. А также проблемы могут возникнуть по причине того, что изображение обладает достаточно большим размером, что увеличит вычисления для однопоточной реализации. Но если этот процесс распределить между потоками, то каждый поток будет отвечать за свою собственную задачу.

2.2 Проектирование многопоточной реализации алгоритмов HOG с SVM

Улучшения алгоритма гистограммы ориентированных градиентов не даст значительного прироста к производительности, в свою способность быстро преобразовывать входной кадр изображения. В свою очередь алгоритм SVM можно распределить между ядрами процессора, что даст весомый

прирост к производительности. В качестве предобработки входного изображения выступают алгоритм HOG. Блок, отвечающий за вычисление гистограммы ориентированных градиентов, включает в себя несколько встроенных функций, которые не будут задействованы при распараллеливании, в свою очередь алгоритм SVM потребуется распределить между потоками для улучшения эффективности. Исходя их этого найдена точка для внедрения технологии распараллеливания. Теперь, когда однопоточная вариация получен и найдена точка внедрения технологии распараллеливания, можно приступить к построению блок-схемы многопоточной реализации (рисунок 29):

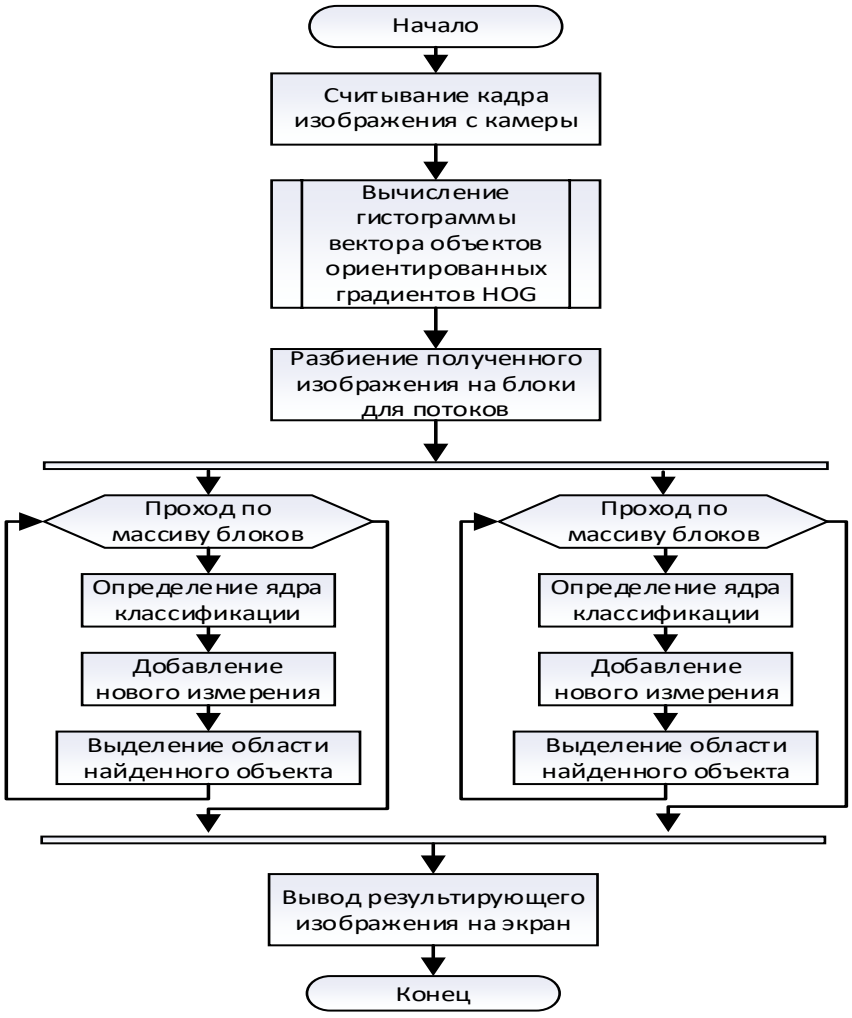

Рисунок 29 – Блок-схема многопоточного варианта применения алгоритмов HOG с SVM

Самой нагруженной областью, приведённой блок схемы, является блок с отвечающий за применение алгоритма SVM и к данному блоку применяется технология распараллеливания, как видно на приведённом рисунке. Блок, отвечающий за предобработку входного кадра изображения с помощью алгоритма HOG, несёт под собой набор последовательных функций, которые работают с входным кадром изображения. В их число входят следующие функции:

- вычисление градиентных изображений;
- вычисление гистограммы градиентов;
- блочная нормализация полученных гистограмм;
- вычисление гистограммы вектора объектов ориентированных градиентов.

Исходя из понимания работы алгоритма SVM, включающего в себя можно несколько вычислительных функций, можно сделать вывод, что данные функции подвержены сильной вычислительной нагрузке и их можно распределить через потоки, что даст прирост к вычислительной скорости и следовательно прирост к скорости работы программы по локализации искомых объектов.

Выше были продемонстрированы однопоточное и многопоточное проектирование реализаций модулей по локализации объектов разного рода на изображении и видео потоке для алгоритма Виолы-Джонса и метода HOG с применением алгоритма SVM. Данный подход с применение многопоточной технологии и пред фильтрации искомого изображения через гистограммы ориентированных градиентов помогут увеличить скорость работы алгоритма.

2.3 Принцип работы SVM с применением многопоточной реализации

Применение многопоточной реализации даст прирост к производительности вычислений при применении алгоритмов Виолы-Джонса и метода HOG в связке с алгоритмом SVM. Принцип работы многопоточной реализации будет приведён ниже (рисунок 30):

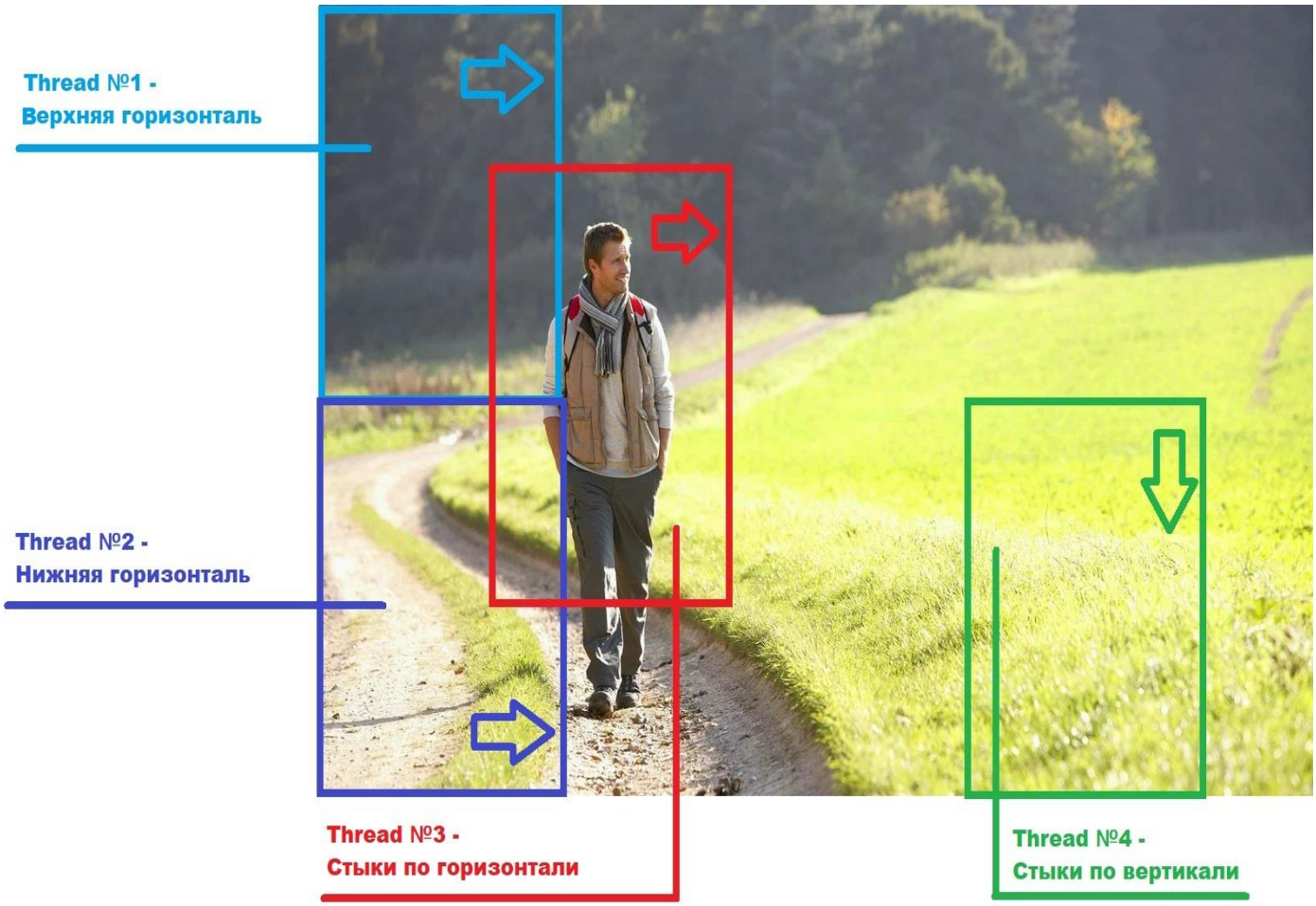

Рисунок 30 – Принцип работы многопоточной реализации алгоритма SVM

На рисунке представлено разбиение входного изображения по различным блокам. Так как система будет тестироваться на процессоре с

четырех ядерной архитектурой, с названием «Intel(R) Core(TM) i5-7600K CPU» с частотой 3.8 GHz, блоки будут привязаны к каждому из четырех ядер.

Как видно на рисунке Thread №1 возьмет на себя задачу по прогону нижней горизонтали от начала и до конца, в свою очередь Thread №2 возьмет верхнюю горизонталь и дойдёт до конца. Возникает проблема, если искомый объект находится на стыках данного распределяя работы, то применяется решение проходить все стыки данного входного изображения, поделённого между потоками Thread №1 и Thread №2. Если посмотреть на вспомогательное ядро Thread №3, то он берёт на себя задачу пройти все стыки по горизонтали, отмеченные зеленной пунктирной линией. В свою очередь имеются стыки по вертикали и данную проблему решает Thread №4, который проходит по вертикальным стыкам, отмеченным пунктирным обозначением синего цвета. Таким образом решается проблема нахождения объекта на пересечениях и оптимизируется работа по распознаванию входного кадра изображения.

Подход обладает гибкостью, в свою очередь, так как для любой архитектуры процессора можно реализовать сценарий выполнения поставленной задачи на локализацию искомого объекта, чтобы все ядра процессора были задействованы в работе, для максимального прироста к производительности. К примеру, если имеется двух ядерная архитектура, то Thread №1 берет на себя две задачи, проверка по верхней горизонтали изображения и проверка всех стыков по горизонтали, в свою очередь Thread №1 проверяет нижнюю горизонталь и стыки на вертикалях.

2.4 Вычисление ускорения работы SVM

Показатель ускорения работы алгоритмов даёт математическое объяснение, насколько применение технологии распараллеливания уместно при использовании в конкретном алгоритме или программе.

Ускорение параллельного алгоритма рассчитывается по формуле (17):

$$s_n = \frac{T_1}{T_n}, \tag{17}$$

где T_n – время вычисления области кода при использовании n потоков;

T_1 – время вычисления области кода при одном потоке.

Исходи из этой формулы вытекают следующие понятия, формулы (18)-(19):

$$T_n < T_1, \tag{18}$$

$$T_n > T_1. \tag{19}$$

Если время выполнения при использовании нескольких потоков меньше времени выполнения при однопоточной реализации, то использование технологии распараллеливания эффективно, если же наоборот, то ресурсные расходы на использование технологии распараллеливания слишком велики [10].

С ускорением тесно связана эффективность работы технологии распараллеливания, формула (20):

$$E_n = \frac{S_n}{n}, \tag{20}$$

Исходя из определения эффективности $E_1 = 1$, теоретически получаем неравенства $S_n \leq n$ и $E_n \leq 1$. Если многопоточная реализация достигает максимального ускорения $(S_n = n)$, то $E_n = 1$. Обычно эффективность спадает при увеличении количества процессов в блоке распараллеливания.

На практике встречается линейное ускорение, обозначенное формулой (21):

$$E_n > 1, \tag{21}$$

Данный вид ускорения ещё называют аномальным, и оно получается путем применения не самого эффективного последовательного алгоритма исходя из существующих, а также может получатся, путем перенаполнением кэш памяти при увеличении рабочих процессов, с большим количеством вычислений [9].

Существует закон Амдала способный теоретически посчитать производительный применения параллельной реализации, и он записывается по формуле (22):

$$S_n = \frac{1}{a + \frac{1-a}{n}}, \tag{22}$$

где S_n –ускорение, представлено кратной величиной при n потоков;

a – доля программного кода, к которой будет применяться параллельная реализация ($a \neq 0$).

Данный закон не учитывая множество факторов накладывает ограничения на максимальную производительность многопоточной реализации. Рассчитаем примерный прирост производительности при использовании многопоточной реализации для алгоритма Виолы-Джонса и HOG с SVM. Основная часть кода, которую можно будет распараллелить заключается в две трети составляющей реализации и получаем, что $a = \frac{1}{3}$, то есть одну треть не будет распараллеливаться. В итоге получаем, что независимо от процессорной архитектуры ускорение составит $S_n < 3$, то есть алгоритмы нельзя будет ускорить больше, чем в три раза.

Существует также закон Густафсона-Бариса оценивающий максимальное ускорение параллельных вычислений исходя из количества задействованных потоков и части последовательных расчетов. Формула данного закона выглядит следующим образом:

$$S_n = n + (1 - n) * a\ , \tag{23}$$

где a – часть последовательного выполнения программы;

n – количество ядер, задействованных для расчетов.

По данному закону известно, что на многопроцессорных системах работа склонна к изменению стратегии решения задачи, исходя из этого снижение общего времени работы программы уступает объёму решаемой задачи [9]. По закону Густафсона-Бариса на четырёхъядерной архитектуре процессора и при последовательной доли кода $a = \frac{1}{3}$, получаем, что максимальное увеличение производительности при применении многопоточного программирования будет составлять $S_n < 3$. И получаем, что ускорить данные алгоритмы теоретически возможно в три раза, но не более.

В результате применения закона Амдаля и закона Густафсона-Бариса удалось вычислить теоретические значения производительности с использованием технологии распараллеливания для алгоритма SVM для дальнейшего сравнения.

3 Программная реализация и тестирование

3.1 Однопоточная реализация

Программная реализация основана на языке Python. Она включает в себя следующие алгоритмы:

– алгоритм Виолы-Джонса;

– алгоритм HOG и SVM.

Программная реализация поддерживает распознавание в реальном времени, а также импортирование видеофайлов формата mp4. Программа обладает гибкой настройкой на выбор источника для распознавания, назначение отдельного алгоритма на выполнение, выбор архитектуры для вычислений CPU или GPU, но это только для алгоритма DNN. Для этой настройки и запуска программы необходимо указать соответствующие ключи и аргументы в командной строке. Рассмотрим их примеры на рисунках ниже (рисунок 31).

```
python face_detection_XXXX.py --video <filename>
```

Рисунок 31 – Команда на выполнение конкретного исполняемого файла

Все исполняемые файлы начинаются на именование «face_detection_», к примеру «face_detection_opencv_haar.py» запускающий алгоритм Виолы-Джонса. Также вторым аргументом идёт ключ «--video», отвечающее за распознавание с видео потока, он требует после себя конкретный путь до видео файла.

```
python face_detection_opencv_dnn.py --video <filename> --device cpu
```

Рисунок 32 – Команда на выполнение с выбором девайса

В данном примере имеется выбор конкретного исполняемого файла, а также представлен ключ «--device», отвечающий за выбор архитектуры вычисления CPU или GPU.

Если необходимо выбрать на исполнение сразу все алгоритмы по распознаванию объектов заложенные в систему, то необходимо использовать следующий вариант запуска (рисунок 33):

```
python run-all.py --video <filename>
```

Рисунок 33 – Команда на выполнение сразу всех алгоритмов в программной реализации

В данном случае ключ «--video» не является обязательным, если его опустить, то исходный поток кадров будет идти с веб камеры компьютера и тем самым мы получим распознавание объектов в реальном времени.

Данная реализация заточена на распознавание нескольких лиц на входном потоке кадров. Рассмотрим реализацию используемых алгоритмов, представленных выше в однопоточном варианте, и начнём с алгоритма Виолы-Джонса (рисунок 34):

```
def detectFaceOpenCVHaar(faceCascade, frame, inHeight=300, inWidth=0):
    frameOpenCVHaar = frame.copy()
    frameHeight = frameOpenCVHaarframeOpenCVHaar.shape[0]
    frameWidth = frameOpenCVHaar.shape[1]
    if not inWidth:
        inWidth = int((frameWidth / frameHeight) * inHeight)

    scaleHeight = frameHeight / inHeight
    scaleWidth = frameWidth / inWidth

    frameOpenCVHaarSmall = cv2.resize(frameOpenCVHaar, (inWidth, inHeight))
    frameGray = cv2.cvtColor(frameOpenCVHaarSmall, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(frameGray)
```

Рисунок 34 – Программный код с применением алгоритма Виолы-Джонса

Функция для вызова алгоритма Виолы-Джонса имеет название «detectFaceOpenCVHaar», в данную функцию передаётся четыре аргумента «faceCascade», «frame», «inHeight», «inWidth». Первый аргумент отвечает за каскад распознаваемого объекта, в данном случае аргумент имеет каскад на локализацию человеческого лица, второй подразумевает кадр изображения, двое других аргументов отвечают за размер окна распознавания. Внутри функции идёт копирование входного кадра, вычисление вспомогательных величин для скользящего окна распознавания, сжатие входного изображения на основе «inHeight», «inWidth», а также применение фильтрации градиент серого для входного изображение, после чего получаем массив областей, где имеется распознаваемый объект. Для наглядности распознанного объекта производится его отрисовка (рисунок 35):

```
bboxes = []
for (x, y, w, h) in faces:
    x1 = x
    y1 = y
    x2 = x + w
    y2 = y + h
    cvRect = [
        int(x1 * scaleWidth),
        int(y1 * scaleHeight),
        int(x2 * scaleWidthscaleWidth),
        int(y2 * scaleHeight),
    ]
    bboxes.append(cvRect)
    cv2.rectangle(
        frameOpenCVHaar,
        (cvRect[0], cvRect[1]),
        (cvRect[2], cvRect[3]),
        (0, 255, 0),
        int(round(frameHeight / 150)),
        4,
    )
return frameOpenCVHaar, bboxes
```

Рисунок 35 – Программный код с отрисовкой результатов работы алгоритма Виолы-Джонса

Объект «cvRect» хранит в себе четыре координаты, соответствующие найденному объекту на изображении, с помощью функции «cv2.rectangle» производится отрисовка прямоугольника на изображении сжатого формата с использованием объекта «cvRect».

Рассмотрим реализацию использования алгоритмов в связке HOG и SVM (рисунок 36):

```
def detectFaceDlibHog(detector, frame, inHeight=300, inWidth=0):

    frameDlibHog = frame.copy()
    frameHeight = frameDlibHog.shape[0]
    frameWidth = frameDlibHog.shape[1]
    if not inWidth:
        inWidth = int((frameWidth / frameHeight) * inHeight)

    scaleHeight = frameHeight / inHeight
    scaleWidth = frameWidth / inWidth

    frameDlibHogSmall = cv2.resize(frameDlibHog, (inWidth, inHeight))

    frameDlibHogSmall = cv2.cvtColor(frameDlibHogSmall, cv2.COLOR_BGR2RGB)
    faceRects = detector(frameDlibHogSmall, 0)
    print(frameWidth, frameHeight, inWidth, inHeight)
```

Рисунок 36 – Программный код с применением алгоритмов HOG и SVM

Первым делом мы копируем входной кадр изображения с помощью функции «copy», далее получаем высоту и ширину входного кадра изображения с помощью функции «shape» и обращении к отдельному объекту массива. Вычисляем масштаб входного кадра, для будущей отрисовки найденного объекта. Далее уменьшаем изображение до рабочего размера,

послле меняем цветовое пространство на RGB с помощью функции «cvtColor». После чего получаем массив найденных объектов с применением HOG и SVM с помощью вызванного объекта «detector». После чего производится выделение найденных объектов на входном кадре изображения (рисунок 37):

```
cvRect = [
    int(faceRect.left() * scaleWidth),
    int(faceRect.top() * scaleHeight),
    int(faceRect.right() * scaleWidth),
    int(faceRect.bottom() * scaleHeight),
]
bboxes.append(cvRect)
cv2.rectangle(
    frameDlibHog,
    (cvRect[0], cvRect[1]),
    (cvRect[2], cvRect[3]),
    (0, 255, 0),
    int(round(frameHeight / 150)),
    4,
)
```

Рисунок 37 – Программный код с отрисовкой результатов работы алгоритмов HOG и SVM

На данном рисунке нижняя часть – функция «rectangle» идентична той, которая используется с алгоритмом Виолы-Джонса, но объект «cvRect» строиться немного другим образом, вызывается объект «faceRect», в котором содержится один из результатов работы алгоритмов. После чего доступ к координатам осуществляется через вложенные функции «left», «top», «right» и «bottom», после чего производится перемножение на величину вычисленного ранее масштаба.

3.2 Многопоточная реализация

Для многопоточной реализации было притяни использовать модуль «multiprocessing». Отличительной особенностью данного модуля является предлагаемые удобные средства параллельного выполнения функций для нескольких входных значений с автоматическим распределением по ядрам процессора, что отлично подходит для распределения областей локализации между потоками. Также немаловажной особенностью является обмен данными между несколькими потоками. Модуль «multiprocessing» поддерживает два канала связи между потоками:

– канал Queues;

– канал Pipes.

Канал Queues является аналогом класса Queues – безопасным для нескольких потоков в различных процессах. Канал Pipes обладает несколькими объектами, которые имеют двунаправленное соединение. С использованием данных каналов можно общаться между потоками. Такой подход избавляет от применения тривиальных синхронизаций, к примеру блокировок.

Также при использовании «multiprocessing» потоки обладают общей памятью, что является возможностью передавать не константные значения с последующей заменой входного аргумента, не опасаясь ошибочных ситуаций при локализации объектов. Далее будет представлена многопоточная реализация с использованием модуля «multiprocessing» (рисунок 38).

```
NUM_CPU = os.cpu_count() - 1

def worker(argc):
  faceCascade = argc[0]
  frame = argc[1]
  result = argc[2]

  frame = splitFrameByThread(frame)
  return detectFaceOpenCVHaar(faceCascade, frame, result, inHeight=300, inWidth=0)

if __name__ == '__main__':
  with multiprocessing.Pool(NUM_CPU) as pool:
  pool.map(worker, argc)
```

Рисунок 38 – Программная реализуя с применением многопоточного модуля «multiprocessing»

Исходя из приведённого листинга видно, что константное значение «NUM_CPU» хранит в себе количество потоков, которые будут использованы для распараллеливания локализации. Функция «os.cpu_count» возвращает количество процессоров, поддерживаемое системой. Функция worker принимает на вход аргументы, которые будут доступны всем процессам, самый главный объектом является результирующая переменная «result», которая будет содержать в себе результат работы всех потоков. Для разделения входного кадра изображения между потоками по вертикали, горизонтали и на стыках используется функция «splitFrameByThred», которая в зависимости от задействованного потока выдаёт определённую область для локализации. После того, как аргументы были получены, они передаются конкретному алгоритму для распознавания. Чтобы задействовать подход

«multiprocessing» необходимо объявить пул потоков с помощью конструкции «multiprocessing.Poll», которая принимает на вход значение распределяемы потоков. После того, как потоки были зарезервированы, необходимо передать рабочее окружение с помощью «poll.map», которая включает в себя используемую функцию и аргументы на вход для локализации.

3.3 Тестирование программного решения

Тестирование будет проводиться с использованием загруженного видеоролика в приложение, а также с использованием встроенной камеры компьютера 1,3 МП (1280x720). В качестве цели распознавания будет использовано лицо человека.

На рисунке 39 будет представлен результат распознавания метода Виолы-Джонса без использования подхода «multiprocessing». За исходные данные будет взята веб-камера компьютера.

Рисунок 39 – Тестирование алгоритма Виолы-Джонса без использования подхода «multiprocessing»

На данном рисунке видно, что алгоритм Вила-Джонса справляется со своей задачей за 25,51 кадр в секунду. Далее будет представлен результат работы алгоритмов HOG с SVM с применением подхода «multiprocessing» (рисунок 40):

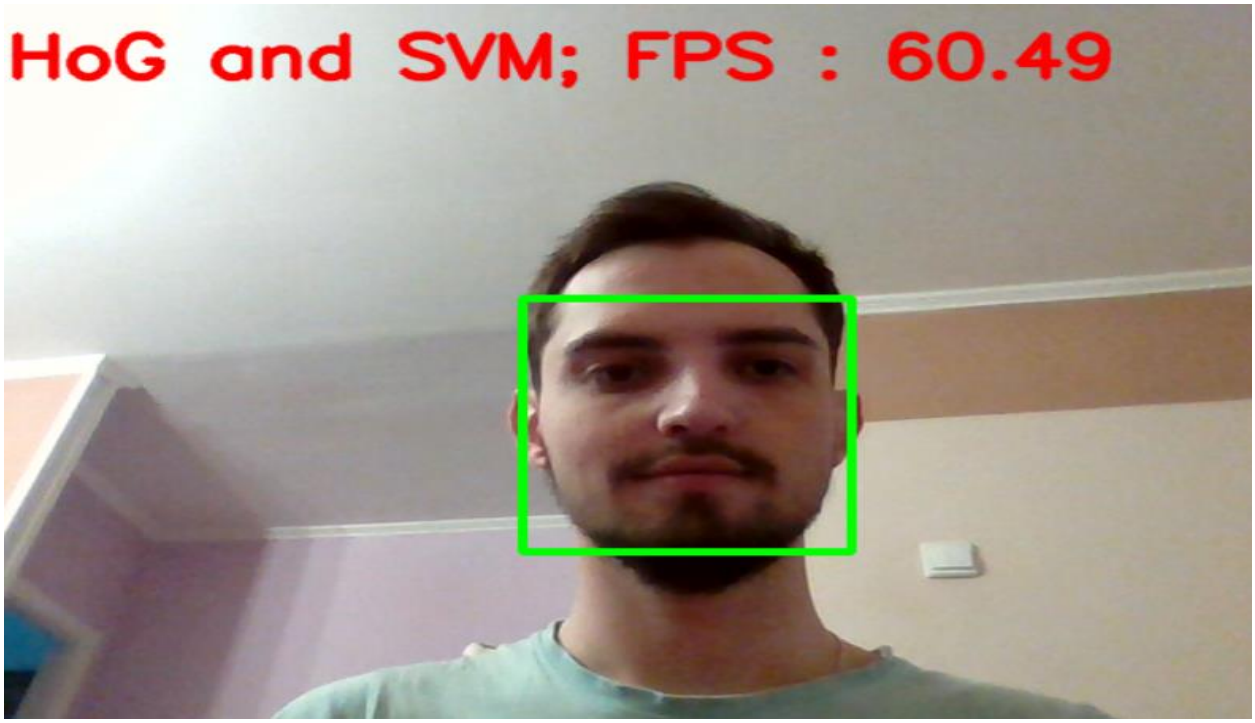

Рисунок 40 – Тестирование алгоритма HOG и SVM с использования подхода «multiprocessing»

На данном рисунке видно, что связка алгоритмов HOG и SVM, с применением подхода «multiprocessing» обрабатывает 60,49 кадров в секунду и такой результат превосходит алгоритм Виолы-Джонса в стандартном варианте реализации.

Далее будет представлена диаграмма результатов работы алгоритмов с использованием подхода «multiprocessing» и без него (рисунок 41):

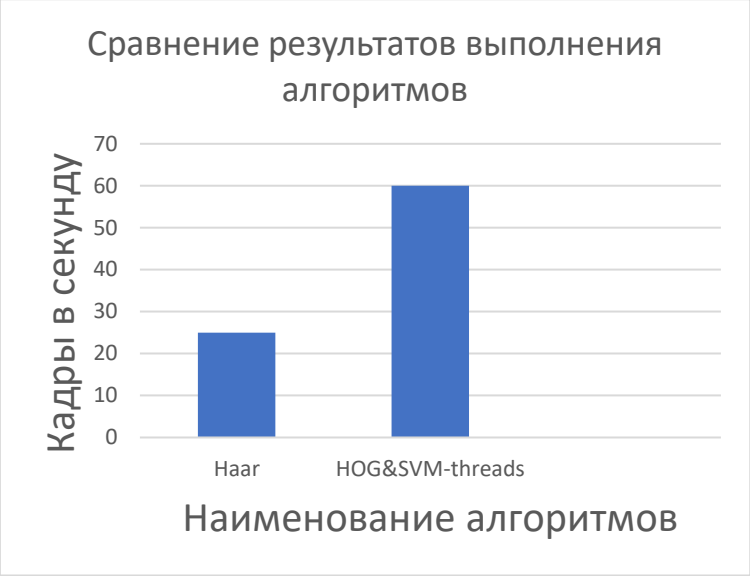

Рисунок 41 – Результаты тестирование алгоритмов

На данном рисунке наглядно представлены результаты тестирования алгоритмов на входном потоке кадров изображения с целью распознавания человеческого лица. В процессе тестирование было задействовано четыре ядра процессора «AMD Ryzen 5 3600» для распараллеливания. Тестирование производилось на загруженном видеоролике, а также с использованием встроенной камеры компьютера. Производилось вычисление среднего значения и округление до целой части числа. По приведённым результатам можно сделать вывод, что алгоритм Виолы-Джонса с результатом в 25,51 кадр в секунду уступает связке алгоритмам HOG и SVM с частотой 60,49 кадров в секунду использованием технологии «multiprocessing» в 2,37 раза.

Исходя из полученных теоретических значений в результате применения закона Амдаля и закона Густафсона-Бариса максимальный прирост в производительности на четырёхъядерной архитектуре составил $S_4 < 3$, что в полной мере удовлетворяет практическому тестированию с результатом 2,37 < 3.

Заключение

В ходе выполнения магистерской диссертации были проанализированы существующие модификации алгоритма Виолы-Джонса, рассмотрены популярные алгоритмы фильтрации изображения, а также логическая обработка результатов фильтрации, были исследованы существующие алгоритмы по локализации объектов на изображении и видеопотоке, а также приведены их достоинства и недостатки. Были рассмотрены вычислительные архитектуры для применения технологий распараллеливания на языке python и выбрана конкретная технология «multiprocessing». Были спроектированы однопоточная блок-схема алгоритма Виолы-Джонса и многопоточная блок-схема алгоритма HOG и SVM с применением технологии «multiprocessing», а также был сформирован подход по распараллеливанию блока локализации алгоритма SVM на архитектуре CPU, а именно логика разбиения входного кадра изображения на области, которые распределяются между потоками. Также были вычислены теоретические показатели прироста производительности с помощью закона Амбала и Густафсона-Бариса для сравнения с результатом тестирования. Была разработана программа для тестирования алгоритма Виолы-Джонса и модифицированного алгоритма SVM с применением HOG фильтрации.

Из полученных результатов в ходе тестирования было выявлено, что алгоритмы в связке HOG и SVM работаю в 2,37 раз быстрее, чем стандартная реализация алгоритма Виолы-Джонса и исходя из теоретических результатов Амбала и Густафсона-Бариса, данный результат удовлетворяет условию $S_4 < 3$, то есть на четырех ядерной архитектуре мы получаем следующее неравенство $2{,}37 < 3$, что удовлетворяет вышеперечисленным законам.

Исходя из данных результатов, можно сделать вывод, что гипотеза о повышении эффективности работы алгоритмов Виолы-Джонса при использовании разработанной модели распределения локализации объектов на изображения между потоками является в полной мере доказанной.

Список используемой литературы

1. Визильтер Ю. В. Обработка и анализ изображений в задачах машинного зрения: Курс лекций и практических занятий. М.: Физматкнига, 2010. – 672 с.

2. Гарсия Г. Б., Исмаэль С.Г., Ноэлия В. Э. Обработка изображений с помощью OpenCV. М.: ДМК Пресс, 2016. 210 с.

3. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. М.: Техносфера, 2012. – 1104 с.

4. Использование GPU для решения задач компьютерного зрения в библиотеке OpenCV [Электронный ресурс] / Режим доступа: http://agora.guru.ru/hpch/files/Using_GPU_for_computer_vision_in_OpenCV_library.pdf

5. Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. – СПб.: Питер, 2016. – 800 с.

6. Клетте Р. Компьютерное зрение. Теория и алгоритмы. М.: ДМК Пресс, 2019. 508 с.

7. Машинное зрение: понятия, задачи и области применения [Электронный ресурс] / Режим доступа: http://rusnauka.com/25_SSN_2009/Informatica/51050.doc.htm

8. Н. Красильников Цифровая обработка 2D- и 3D-изображений / Красильников Н.: Отдельное издание. - БХВ-Петербург, 2016. - 608 с.

9. Обнаружение пешеходов HOG и SVM [Электронный ресурс] / Режим доступа: https://russianblogs.com/article/1298987091/

10. Системы компьютерного зрения: современные задачи и методы [Электронный ресурс] / Режим доступа: http://controleng.ru/innovatsii/sistemy-komp-yuternogo-zreniya-sovremenny-e-zadachi-metody/.

11. Что такое машинное зрение и чем оно отличается от человеческого? [Электронный ресурс] / Режим доступа:

https://meduza.io/feature/2019/03/30/chto-takoe-mashinnoe-zrenie-i-chem-ono-otlichaetsya-ot-chelovecheskogo-seychas-ob-yasnim-ponyatno

12. Beyerer J. Automated Visual Inspection: Theory, Practice and Applications / J. Beyerer, P. Fernando, F. Christian. – Springer Berlin Heidelberg, 2016. – 798 p.

13. Christopher M. Pattern Recognition and Machine Learning (Information Science and Statistics). – Springer 2006. – 738 p.

14. Efficient face detection algorithm: using Viola Jones method [Электронный ресурс] / Режим доступа. URL: https://www.codeproject.com/Articles/85113/Efficient-Face-Detection-Algorithm-using-Viola-Jon

15. Faster RCNN for Face Detection on a FACENET Model [Электронный ресурс] / Режим доступа: https://link.springer.com/chapter/10.1007/978-981-16-2794-1_25

16. Francois C. Deep Learning with Python 1st Edition. – Manning, 2017. – 384 p.

17. Faster and better: a machine learning approach to corner detection [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/0810.2434.pdf

18. Feature detection and description [Электронный ресурс] / Режим доступа: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

19. Getting Started with Image Preprocessing in Python [Электронный ресурс] / Режим доступа: https://www.section.io/engineering-education/image-preprocessing-in-python

20. Introduction To Feature Detection And Matching [Электронный ресурс] / Режим доступа: https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d

21. John T. Mastering Large Datasets with Python: Parallelize and Distribute Your Python Code 1st Edition. – Manning, 2020. – 312 p.

22. Lutz M. Learning Python, 5th Edition Fifth Edition. – O'Reilly Media; Fifth edition, 2013. – 1648 p.

23. Multiprocessing — Process-based parallelism [Электронный ресурс] / Режим доступа: https://docs.python.org/3/library/multiprocessing.html

24. OpenCV: Cascade Classifier [Электрон. ресурс]: Документация URL: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (дата обращения: 10.02.2020).

25. OpenCV-Python Tutorials [Электронный ресурс] / Режим доступа: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

26. Running Python script on GPU [Электронный ресурс] / Режим доступа: https://www.geeksforgeeks.org/running-python-script-on-gpu

27. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [Электронный ресурс] / Режим доступа: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

28. Training SVM classifier with HOG features [Электронный ресурс] / Режим доступа: https://www.kaggle.com/code/manikg/training-svm-classifier-with-hog-features/notebook

29. The Computer Vision Pipeline, Part 3: image preprocessing [Электронный ресурс] / Режим доступа: https://freecontent.manning.com/the-computer-vision-pipeline-part-3-image-preprocessing/

30. Vehicle Detection with HOG and Linear SVM [Электронный ресурс] / Режим доступа: https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a