ВВЕДЕНИЕ

Зерно – основная продукция сельского хозяйства. Из зерна вырабатываются такие продукты питания, как мука, крупы, макаронные и хлебобулочные изделия. Зерно необходимо для производства корма в животноводстве и птицеводстве. Также оно используется для изготовления крахмала, спирта и других продуктов. Ввиду его широкого применения, качеству зерна уделяется высокое внимание.

Качество зерна – важный и обязательный объект государственного планирования и контроля. Стандартизация лежит в основе государственной системы управления качеством зерна. Cоздание однородных партий зерна и обеспечение его сохранности требовали строгого нормирования качества, поэтому зерно стало одним из первых объектов стандартизации.

Оценка качества зерна производится с использованием двух показателей: общие показатели – обязательные, определяемые в любой партии зерна всех культур признаки свежести (внешний вид, цвет, запах, вкус), зараженность зерна вредителями, влажность и засоренность; специальные, или целевые, — показатели качества, определяемые в партии зерна отдельных культур, используемых на конкретные цели (плёнчатость и выход чистого зерна, стекловидность, жизнеспособность, натура и другие).

Анализ качества зерна осуществляется в лабораториях предприятий, осуществляющих приём, хранение и переработку культуры. Качество партии оценивают по результатам лабораторного анализа средней пробы, которые распространяются на всю партию. Лаборатории оснащены приборами и лабораторным оборудованием, располагая которыми, специалисты могут производить оценку зерна на всех стадиях его обработки. Анализировать качество можно стандартными методами или альтернативными экспресс-методами. Существующие подходы затруднены большим количеством ручного труда и малой долей автоматизации, что увеличивает время затраченное на работу, а также вероятность ошибки, из-за присутствия человеческого фактора. Особенно ярко это наблюдается в методах оценивающих физическое свойство зерна и зерновой массы (форма, линейные размеры, содержание сорной и зерновой примесей).

Для данной работы разрабатывается экспресс-метод оценки качества зерна, в основе которых лежит компьютерный анализ цифрового изображения сырья. Данный подход позволяет не только оперативно получить геометрические параметры, но также распознать общий состав зерновой массы поступающей на предприятие, оценить степень интенсивности воздействия на зерно в процессе переработки, как по отдельным этапам, так и в целом показатели качества готовой продукции.

После анализа предметной области была сформирована цель данного дипломного проекта – разработать программное средство анализа качества зерновых культур для широкого класса изображений.

**1** ОБЗОР ЛИТЕРАТУРЫ

Первоначально необходимо отметить, что с учетом всевозможных достоинств и недостатков известных технологий, сред разработок, операционных систем, были выбраны следующие компоненты и технологии: для написания программного кода язык программирования C++ вместе со средой разработки Qr Creator, выбор был сделан в их пользу за скорость работы и кроссплатформенность. В качестве сторонней библиотеки была взята библиотека алгоритмов компьютерного зрения с открытым кодом – OpenCV.

**1.1** Обзораналогов

В связи с увеличением потребностей пищевой промышленности в объемах зерновых культур, встает вопрос о хранении и обработке. Семена высокого качества лучше сохраняются и на это требуется меньшая затрата средств предприятия. Как известно, чем выше качество конечной продукции, тем больше внимание уделяется предварительному анализу ингредиентов.

Первым шагом на пути к улучшению качества является повсеместный контроль за качеством зерна [1]. Данная практика введена во многих странах мира и является стандартом. У зерна можно выделить ряд обязательных признаков зрелости и свежести (внешний вид, вкус, запах), зараженность вредителями, механические повреждения, влажности, а так же содержание сорной и зерновой примеси (см. рисунок 1.1) [2].



Рисунок 1.1 – Пример изображений

Работа ставит цель помочь пищевой промышленности в процессе анализа и подготовки зерна. Разработанное средство помогает обработать изображения культур и сделать вывод о его качественных и количественных показателях.

В данной области есть ряд продуктов, которые выполняют аналогичную функцию. Все они имеют различные характеристики и параметры, обзор позволит определить какие параметры не реализованы на данный момент. Общий обзор аналогов представлен на вводном плакате [3].

# **1.1.1** Инфракрасный экспресс-анализ

Сейчас на рынке присутствует несколько моделей от разных производителей: GRANOLYSER, INFRAMATIC-9500, INFRANEO, ИНФРАСКАНЕР-105 [4]. Данные модели используют для анализа зерна длинноволновую ИК-область спектра. Приведенное оборудование способно производить подсчет количества зерен, определять влажность, белок, масло, клейковина. Плюсами являются: скорость, возможность анализа качественных признаков. Минусами – анализирует только на основе ИК-области спектра, не может различать разные типы культур, требует дорогостоящего оборудования [5].

**1.1.2** Рентгенографический анализ

Данный метод основывается на автоматическом анализе графических файлов рентгенобразов семян зерновых культур. Метод позволяет визуализировать объекты малого размера для выявления структурных дефектов внутренней структуры. Определяются такие характеристики: трещиноватость, механические повреждения признаки заселенности насекомыми, поражения сосущими насекомыми и количество щуплого зерна [6]. Плюсы данного подхода: возможность анализа внутренней структуры, высокая точность. Минусы: специализированное оборудование, работа только с одним типом зерна за один замер [7].

**1.2** Методы применяемые для предобработки изображения

Методы цифровой обработки позволяют преобразовать изображение для дальнейшего анализа. Классическая схема разработки системы распознавания (классификации) состоит из нескольких задач. Первая, выбор тех признаков, которые достаточно полно описывают образ. Вторая, выбрать из признаков те, которые позволят отделить объекты одного класса от объектов другого. Третья, выбор такого классификатора, который поможет решить данную задачу с максимальной эффективностью. Четвертая, проанализировать результаты классификации и предоставить пользователю вывод об этом. В итоге весь алгоритм сводится к шагам (см. рисунок 1.2):

* предварительная обработка;
* вычисление признаков;
* принятие решения;



Рисунок 1.2 – Классическая схема системы классификации

**1.2.1** Цветовые модели

Цветовая модель – математическая модель описания представления цветов в виде кортежей чисел (обычно из трех, реже – четырех значений), называемых цветовыми компонентами. Все возможные значения цветов, задаваемые моделью, определяют цветовое пространство.

RGB(Red, Green, Blue) – аддитивная цветовая модель, которая описывает способ синтеза цвета для цветовоспроизведения. Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза.

CMYK(Cyan, Magenta, Yellow, Key color) – схема формирования цвета, основанная на методе получения цвета путем вычитания из белого света отдельных спектральных составляющих. Эта модель обладает сравнительно меньшим с RGB цветовым охватом.

Lab – схема в которой изменение цвета более линейно с точки зрения человеческого восприятия, т.е. одинаковое изменение значений координат цвета в разных областях цветового пространства производило одинаковое ощущение изменения цвета.

HSV(Hue, Saturation, Value) – модель является нелинейным преобразование RGB [8].

**1.2.2** Математическая морфология

Математическая морфология – техника и теория обработки и анализа геометрических структур. Основана на теории множеств, топологии и случайных функциях. Существует набор базовых операций:

Перенос – операция сдвигающая все пиксели множества на заданное расстояние (см. рисунок 1.3). Вектор переноса может задаваться в виде пары, где первое значение это компонент вектора переноса в направлении строк, а второй компонент – в направлении столбцов изображения.

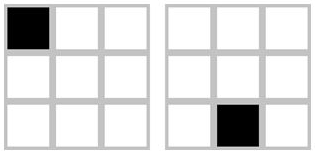


Рисунок 1.3 – Перенос

Наращивание – операция над бинарным изображением А структурным элементом В задается выражением 1.1:

(1.1)

Структурный элемент В применяется ко всем пикселям бинарного изображения. Каждый раз, когда начало координат структурного элемента совмещается с единичным пикселем, ко всему элементу применяется операция переноса и последующее логическое сложение с пикселями бинарного изображения (см. рисунок 1.4).

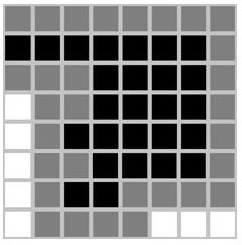


Рисунок 1.4 – Наращивание

Эрозия – это операция над изображением А структурным элементом В задается выражением 1.2:

(1.2)

При выполнении операции эрозии структурный элемент так же проходит по всем пикселям изображения. В позициях, где каждый единичный пиксель структурного элемента совпадает с единичным пикселем бинарного изображения, выполняется логическое сложение центрального пикселя структурного элемента с соответствующим ему пикселем выходного изображения. После применения данной операции все объекты, меньшие чем структурный элемент, исчезают (см. рисунок 1.5).

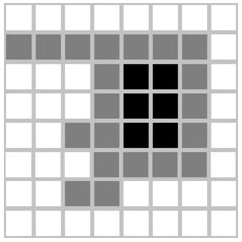


Рисунок 1.5 – Эрозия

Замыкание – операция над бинарным изображением А структурным элементом В задается выражением 1.3:

(1.3)

Операция замыкания убирает небольшие внутренние «дырки» и углубления на краях области.

Размыкание – операция над бинарным изображением А структурным элементом В задается выражением 1.4:

(1.4)

Операция эрозия удаляет малые объекты и различные шумы, но при этом оставшиеся объекты уменьшаются в размерах, избежать этого помогает применение операции наращивания с тем же структурным элементом. Размыкание подходит так же для удаление линий, толщина которых меньше, чем диаметр структурного элемента. После её применения контуры объектов становятся более гладкими [9].

**1.2.3** Фильтрация изображений

Цифровые изображения подвержены воздействию шумов, которые появляются от способа получения, технологий передачи и методов оцифровки. Процесс при котором на изображении устраняются различные виды шумов, называется фильтрацией.

При осуществлении фильтрации яркостные характеристики каждой точки изображения заменяются значение грудой точки признанной в наименьшей степени искаженной. Существует пространственная и частотная фильтрация.

В основу частотных методов лежит идея Фурье преобразования, который заключается в представлении исходной функции в виде суммы тригонометрических функций различных частот, умноженных на заданные коэффициенты.

Пространственные методы применяются к растровым изображениям. Их принцип заключается в применении специальных операторов к каждой точке на изображении. Операторам могут выступать прямоугольные или квадратные матрицы называемые ядрами или окнами.

Линейная оконная фильтрация в пространственной области заключается в вычислении линейной комбинации значений яркости пикселей в окне фильтрации с коэффициентами матрицы весов фильтра, которую так же называют ядром. В качестве линейного сглаживающего фильтра используется усредняющий фильтр, у которого выходное значение является среднее значение по окрестности ядра фильтра. Пример данного подхода является винеровский фильтр. Он является адаптивным линейным фильтром. Принцип работы заключается в том, что если значение среднеквадратичного отклонения интенсивностей пикселей в данной локальной области большое, то фильтр выполняет небольшое сглаживание, а при меньшем отклонении, наоборот, область сглаживания больше.

Нелинейные пространственные методы подобны по принципу работу линейным фильтрам. Операции зависят от значений элементов матрицы изображения, которые находятся в анализируемой окрестности. Примером данного метода является медианный фильтр. При его использовании значения пикселей получаются из усредненного значения точек соответствующей области. Для задач устранения шума этот фильтр является более подходящим, чем усреднение, так как приводит к меньшим искажениям границ области.

Так же существуют фильтры для решения задач сегментации. Сегментация нужна для разделения изображения на части, для осуществления дальнейшего анализа, и для изменение формы описания элементов изображения, что позволяет представить точки как высокоуровневые структуры, которые обеспечат эффективность дальнейшего анализа изображения. Существует множество операторов для решения данной задачи: перекрестный оператор Робертса, оператор Превитта, оператор Собеля, Canny. Принцип работы таких методов основан на разности яркости элементов и фона изображения (см. рисунок 1.6) [10].

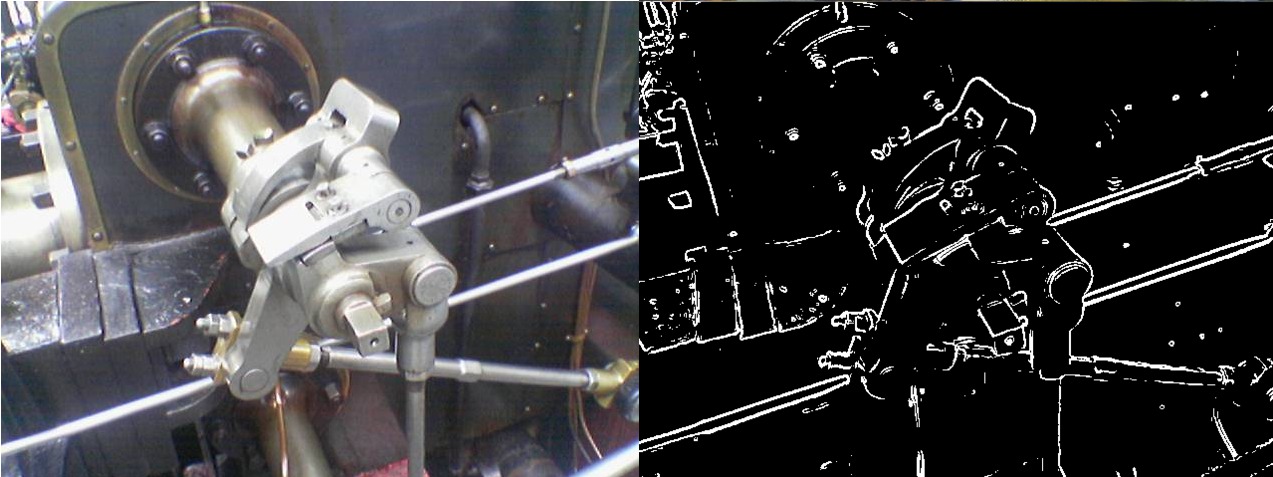


Рисунок 1.6 – Выделение контуров

**1.3** Задача разделения объектов изображения на классы

Для решения данной проблемы существуют классификаторы. Задача классификаторов состоит в том, что отнести объект к одному из уже определенных классов на основании его формализованных признаков. Каждый объект в это задаче представляется как вектор в N-мерном пространстве, где каждое измерение представляет собой описание одного признака объекта [11].

LDA (Латентное размещение Дирихле) – это порождающая модель, которая позволяет объяснять результаты наблюдений с помощью неявных переменных. Неявные переменные – это переменные, которые нельзя изменить в явном виде, а могут быть только выведены через математические модели с использованием наблюдаемых переменных.

KNN (метод k ближайших соседей) – это метрический алгоритм, который позволяет автоматически классифицировать объекты. Является одним из самых простых алгоритмов. Главный принцип метода ближайших соседей это то, что объект относится к тому классу, который является наиболее распространенным среди соседей данного элемента. Соседи выбираются исходя из множества объектов, классы которых уже были определены. Далее исходя из значения k вычисляется, какой класс наиболее многочислен среди них, на основании этого и делается вывод о принадлежности объекта к классу.

SVM (метод опорных векторов) – набор схожих алгоритмов обучения с учителем. Особым свойством метода является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора. Основной идеей является перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве.

Искусственные нейронные сети – это параллельно распределенный процессор, который обладает способностью к сохранению и репрезентации опытного знания. Знание в нейронной сети приобретается сетью в процессе обучения. Для сохранения знания используются силы межнейронных соединений, также называемых синаптическими соединениями (см. рисунок 1.7). Обучением называется процесс выбора параметров сети, при котором сеть наилучшим образом справляется с поставленной задачей.

Современные нейронные сети демонстрируют такие ценные свойства:

* обучаемость;
* способность к обобщению;
* способность к абстрагированию;

К задачам, которые успешно решают нейронные сети относят:

* распознавание зрительных образов;
* ассоциативный поиск информации и создание ассоциативных моделей;
* создание моделей и различных нелинейных и трудно описываемых математических систем;
* системы регулирования с предсказаниями и управления;
* принятие решений и диагностика, исключающие логический вывод;

По архитектуре нейронные сети делятся:

* персептронные сети с прямыми связями;
* самоорганизующиеся нейронные сети;
* нейронные сети с обратными связями;

Данный подход позволяет решать множество задач, как и каждый описанный выше метод искусственные нейронные сети обладают рядом преимуществ и недостатков. К преимуществам можно отнести:

* решение задач при неизвестных закономерностях;
* устойчивость к шумам во входных данных;
* потенциальное сверхвысокое быстродействие;
* адаптирование к изменениям окружающей среды;

Недостатками можно назвать:

* в ряде случаев обучение приводит к тупиковым ситуациям;
* требуется выполнение многоцикловой настройки для построения модели;
* поведение обученной нейронной сети не всегда однозначно предсказуемо;

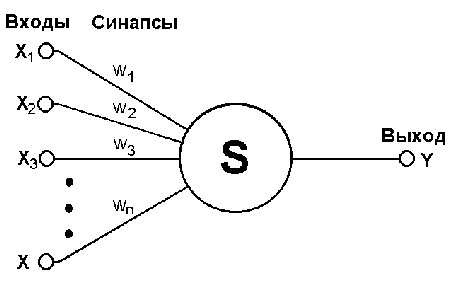


Рисунок 1.7 – Структура нейронной сети

# **2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После анализа предметной области задачи дипломного проекта были определены цели разработки программного средства:

* разработать приложение для анализа зерновой культуры на основе предоставляемых изображений;
* предоставить пользователю возможность производить предобработку исходного изображения используя предлагаемые инструменты (цветовые системы, изменение контраста и яркости, морфологические операции, цифровые фильтры);
* обеспечить удобный и понятный интерфейс пользователя;
* предоставить возможность пользователю выбирать данные для обучения классификатора;
* обеспечить полной и понятной документацией об архитектуре и возможностях программного средства;

Разрабатываемое программное средство должно поддерживать следующие функции:

* ввод изображений для анализа;
* предварительная обработка и фильтрация изображений;
* анализ зерна на содержание примесей других культур либо мусорных примесей;
* отображение результатов в виде отчетов;

**2.1** Структура программного средства

В соответствии с методологией системного подхода в разработке архитектуры, программное средство разбивается на совокупность сущностей, представленных на структурной схеме (см. чертеж ГУИР.400201.035 С1).

На каждый блок в программном средстве возлагается определенная задача. Кроме того каждый блок так или иначе связан с другими блоками, чтобы обеспечить работоспособность всего приложения в целом. Данная связь реализуется посредством обмена данными между блоками.

Блок регистрации изображений – как правило, это аппаратная часть, которая либо производит съемку самих изображений, либо принимает их по сети для дальнейшей передачи в приложение.

База изображений – совокупность изображений для анализа. Изображения могут различаться по качеству, разрешению, нейтральности относительно семян заднего фона. От этих параметров напрямую зависит качество обработки и анализа.

Модуль интерфейса пользователя – разновидность интерфейсов, в котором одна из сторон это человек, а другая компьютер. Модуль предоставляет средства для коммуникации пользователя с программой. Качественная разработка данного блока является важной частью процесса создания программного средства, так как пользователю должно быть интуитивно понятно как им пользоваться.

Модуль выбора цветовой системы – блок, позволяющий изменить цветовую систему изображение. Использую данный подход можно разложить изображение на каналы с другими параметрами и выбрать тот, который лучше отвечает требуемой задаче.

Модуль улучшенияконтраста и яркости – блок, отвечающий за предварительную обработку изображения. Данный модуль используется при недостаточном контрасте или яркости объектов изображения для дальнейшей качественной обработки. Используется алгоритм гамма-коррекции.

Модуль бинаризации – модуль, отвечающий за перевод изображения в бинарный вид, где будут присутствовать только два вида цвета: черный и белый. Данная операция является промежуточной, но вместе с тем обязательной для дальнейшего анализа. Для выполнения данной операции используется на выбор либо алгоритм адаптивной бинаризации, либо метод Оцу.

Модуль алгоритмов фильтрации – блок, позволяющий дообработать изображение после бинаризации. Так как на изображении часто присутствует шум, применяя цифровые фильтры в разных комбинациях, можно достичь нужной чистоты снимка, что является залогом правильной кластеризации, так как шум будет определен далее как объект. В данном блоке был реализован медианный фильтр.

Модуль вычисления геометрических признаков – модуль, относящийся непосредственно к анализу зерна. Модуль, вычисляющий признаки объектов, основанных на геометрических аспектах объекта.

Модуль вычисления яркостных характеристик – модуль, относящийся непосредственно к анализу зерна. Отвечает за исследование яркостных и цветовых характеристик зерновой смеси.

Модуль вычисления текстурных характеристик – модуль, относящийся непосредственно к анализу зерна. Отвечает за вычисление текстурных характеристик зерновой культуры.

Модуль принятия решений – модуль, обеспечивающий основную задачу программного средства – произвести анализ изображения. Блок производит процесс выбора объектов, которые являются эталонными, с помощью пользователя. Далее производится принятие решение о принадлежности объекта к одному из кластеров с помощью классификатора, на основе признаков посчитанных в предыдущих блоках. Классификатором был выбран метод опорных векторов.

Модуль формирования отчетов – модуль, предоставляющий визуализацию принятого программным средством решения. Так же производится анализ качества классификации и построение ROC-кривой, которая является графическим представление качества анализа.

В совокупности, все вышеописанные блоки позволяют решить поставленную задачу с наибольшей эффективностью и полнотой.

Обобщенная структурная схема программного средства изображена на рисунке 2.1.

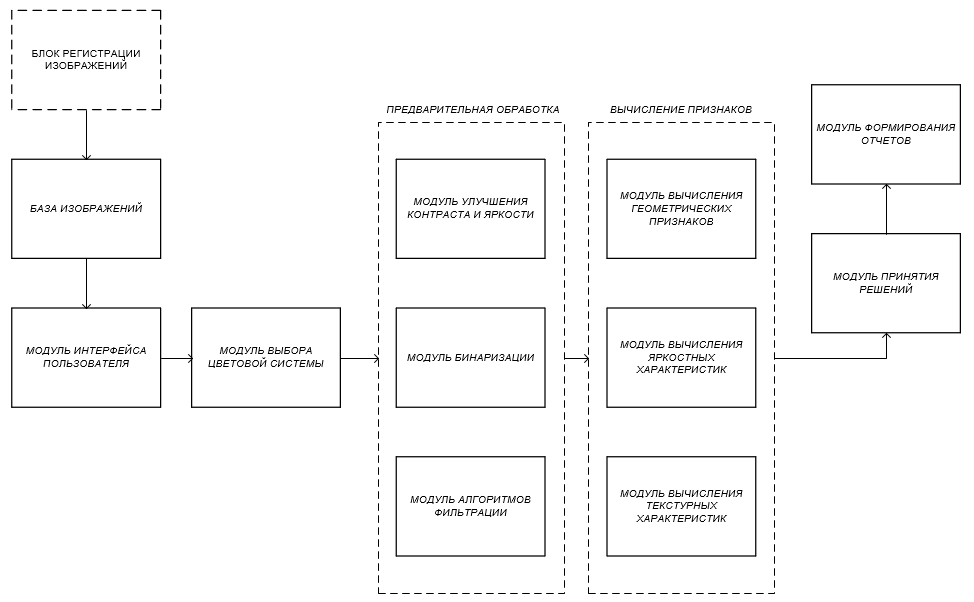


Рисунок 2.1 – Обобщенная структурная схема

**2.2** Средства разработки

Выбор основных технологий является ключевым моментом при подготовке к созданию программного обеспечения, ведь правильный выбор позволит не только упростить процесс проектирования и разработки, но и облегчить эксплуатацию программного продукта.

При выборе языка программирования, на котором будет реализован программное средство, внимание было обращено на такие факторы, как удобство работы, целесообразность использования для решения поставленной задачи, наличия стандарта и документации, поддержка основных операционных систем. На основе этих признаков был выбран С++. Данный язык является компилируемым и статически типизированным. С++ поддерживает такие парадигмы программирования, как процедурное и объектно-ориентированное программирование. Как объектно-ориентированный язык, он оперирует понятиями наследование, инкапсуляция и полиморфизм [12]. Так же обеспечивается модульность, раздельная компиляция, обработка исключений, абстракция данных, объявление типов объектов, виртуальные фукции. К плюсам данного языка можно отнести наличие стандартной библиотеки, которая включает, в том числе, общеупотребительные контейнеры и алгоритмы. Так же С++ является в достаточной степени низкоуровневым, для объектно-ориентированных языков, это позволяет с помощью него решать задачи в которых требуется высокое быстродействие (обработка цифровых изображений и видео, сложные поисковые алгоритмы, сетевые технологии). Спектр задач для него очень широкий, что обеспечило ему огромную популярности среди программистов, на протяжении вот уже нескольких десятков лет [13].

Для разработки интерфейса была выбрана кроссплатформенная свободная IDE – Qt Creator. Она поставляется с фреймворком Qt. Qt Creator позволяет использовать все преимущества объектно-ориентированного подхода, позволяет осуществлять повторное использование кода с большей эффективностью, нежели при применении так называемого модульного программирования. Немаловажным также является использование высокопроизводительных компиляторов и стремление использовать скомпилированный код. Известно, что последний обладает существенно более высокой производительностью, чем код интерпретируемый. Так же Qt Creator обеспечивает визуальными средствами проектирования пользовательских интерфейсов, что позволяет ускорить работу над проектами, облегчить повторное использование кода и в определенной степени привлечь к созданию приложений начинающих программистов. Так же к достоинствам выбранной технологии можно отнести ее кроссплатформенность, это облегчает разработку программных средств, которые планируется использовать на разных операционных системах. В Qt Creator реализовано автодополнение, подсветка кода, задание стиля выравнивания, отступов и постановка скобок, все это облегчает разработку и снимает с программиста часть рутинной работы [14]. Логотип технологии представлен на рисунке 2.2.



Рисунок 2.2 – Логотип технологии Qt

В качестве сторонней библиотеки была выбрана библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом – OpenCV. Данный решение было принято после анализа имеющихся аналогов на предмет соответствия требованиям: высокое быстродействие, большое количество готовых решений алгоритмов, кроссплатформенность и доступная документация. OpenCV поддерживается для таких языков программирования, как С++, Python, Java, это делает библиотеку доступным средством разработки для большего количества программистов. Также библиотека имеет ускорение на уровне аппаратуры [15]. Логотип библиотеки представлен на рисунке 2.3.

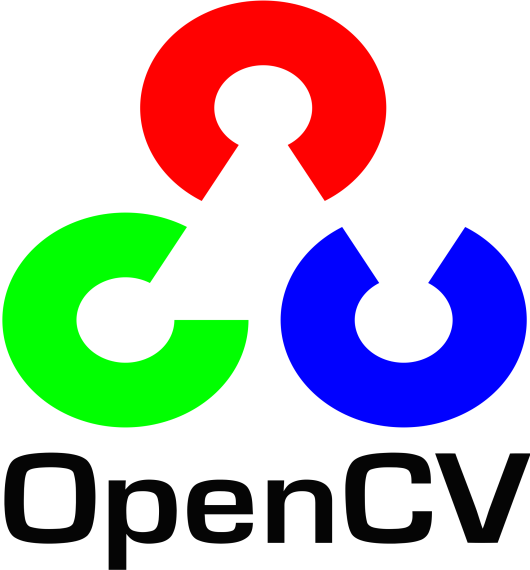


Рисунок 2.3 – Логотип библиотеки OpenCV.

Выбранные инструменты позволяют в полной мере и с высокой эффективностью решить все поставленные задачи данной дипломной работы.

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование программы. Для этого проведем анализ всех классов, которые входят в состав кода программы, и рассмотрим назначение всех методов, свойств и переменных класса.

### 3.1 Класс ColorSystemConverter

Класс обеспечивает работу с цветовыми системами. Осуществляет перевод из одной системы в другую.

Методы:

* static vector<Mat> rgb2cmyk(Mat& img, std::vector<Mat>& cmyk) – метод, принимающий исходною матрицу изображения и ссылку на выходной буфер, переводит изображение из цветовой системы RGB в CMYK путем разложения исходного изображение на отдельные RGB каналы и проводя определенные вычисления над ними сформировать каналы CMYK, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* static vector<Mat> rgb2hsv(Mat img, vector<cv::Mat>& hsvVector) – метод принимает изображение требующее перевода в другую цветовую систему и выходной буфер, производит перевод исходного изображения из RGB в HSV, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* static vector<Mat> rgb2hls(Mat img, vector<cv::Mat>& hlsVector) – метод принимает изображение требующее перевода в другую цветовую систему и выходной буфер, производит перевод исходного изображения из RGB в HLS, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* static vector<Mat> rgb2lab(Mat img, vector<cv::Mat>& labVector) – метод принимает изображение требующее перевода в другую цветовую систему и выходной буфер, производит перевод исходного изображения из RGB в Lab, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* static vector<Mat> showChannels(Mat inputImage, Mat channel[], std::string labels[], double scalar[][3], int convertBack, bool CMYK) – метод, принимающий матрицу изображения, матрицы каналов цветовой системы, цвета для раскраски черно-белые изображения каналов цветовой системы, производит раскраску каналов в цвета для более удобного восприятия человеческим зрением, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* static void cmyk2rgb(const Mat & c ,const Mat & m, const Mat & y, const Mat & k, Mat & rgb ) – метод принимает четыре канала: Cyan, Magenta, Yellow и Key color. Производит перевод разделенных каналов цветовой системы CMYK в систему RGB, метод является статическим и может вызываться от класса, а не от его экземпляра, что упрощает использование;
* explicit ColorSystemConverter(QObject \*parent = 0) – метод принимает ссылку на объект класс предка, является конструкторам, выполняет инициализацию класса. Диаграмма последовательностей метода изображена на рисунке 3.1;

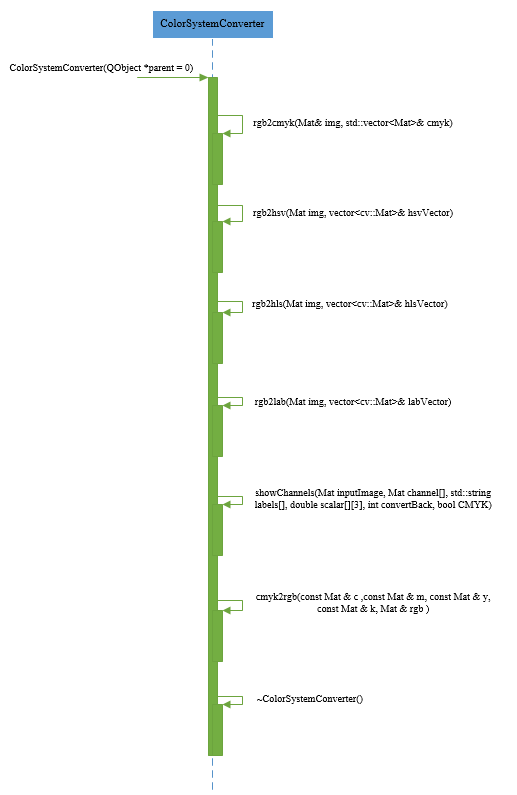


Рисунок 3.1 – Диаграмма последовательности метода ColorSystemConverter()

**3.2** Класс MainWindow

Данный класс является основным классом в проекте, является центром координирующим работу всего приложения. Реализует методы по обработке изображения такие, как цифровые фильтры и морфологические операции, методы по выделению объектов, методы по предоставлению отчетов о проделанной работе. Реализует программную часть интерфейса с пользователем. Так же в нем реализованы методы для работы с другими классами, которые реализуют оставшийся необходимый функционал.

Поля:

* CSvector – сохраняет каналы выбранной цветовой системы, является полем типа vector<Mat>;
* vector<Mat> CSvectorColored – содержит окрашенные в цвета каналы выбранной цветовой системы, является полем типа vector<Mat>;
* lastColorSystem – содержит последнюю выбранную цветовую систему, является полем типа enum ColorSystem;
* matsrc – поле актуальной версии изображения, является полем типа Mat;
* firstImage – поле содержащее начальное изображение, поле типа Mat;
* imgStack – содержит стэк для возможности восстановления предыдущего изображения после неудачного использования цифрового фильтра или морфологической операции, является полем типа QVector<Mat>;
* imgBinaryStack – содержит стэк для возможности восстановления предыдущего изображения после неудачного использования бинаризации, является полем типа QVector< Mat >;
* seedVect – поле содержит все объекты зерен которые были найдены на изображении, является типом QVector<Seed>;
* binarizated – поля требуется для проверки было ли бинаризировано уже изображение, является полем типа bool;
* fileName – содержит имя открытого файла, является полем типа QString;

Методы:

* void OpenPicture() – метод открывает изображение выбранное в диалоге, данное изображение будет в дальнейшем использоваться как исходные данные для алгоритмов;
* void ToGrayScale() – метод переводит изображение в полутон, данная операция является обязательной, так как в дальнейшем будет проводиться бинаризация, а она, в свою очередь, может быть выполнена только на полутоновом изображении;
* void ChangeBrightness() – метод позволяющий произвести коррекцию яркости предоставленного исходного изображения, чтобы улучшить точность выделения объектов;
* void ChangeContrast() – метод позволяющий провести гамма-коррекцию изображение, чтобы, в дальнейшем, улучшить точность выделения объектов;
* void AdaptiveThreshold() – проведение адаптивной бинаризации изображения, метод основан на подходе разбиения изображения на две области, одна из которых содержит все пикселы со значением ниже некоторого порога, а другая содержит все пикселы со значением выше этого порога. Порог выбирается на основе использования локальной и глобальной гистограммы;
* void AdaptiveThresholdOtsu() – метод производит адаптивную бинаризацию Оцу изображения, алгоритм, реализованный в данном методе, позволяет разделить пикселы на два класса: полезные и фоновые, рассчитывая такой порог, чтобы внутриклассовая дисперсия была минимальной;
* void MedianFilter() – метод реализует медианный цифровой фильтр. Данный метод сортирует значения отсчетов внутри окна фильтра, размер окна выбирается пользователем на интерфейсе, в порядке возрастания. Значение, находящееся в середине упорядоченного списка, поступает на выход фильтра. В случае четного числа отсчетов в окне выходное значение фильтра равно среднему значению двух отсчетов в середине упорядоченного списка;
* void DenyFilter() – данный метод производит отмену последнего морфологического преобразования или цифрового фильтра;
* void ErodeFilter() – метод реализует морфологическую операцию эрозии бинаризированного изображения. Операция основана на том что структурный элемент проходит по всем пикселам изображения. Если в некоторой позиции каждый единичный пиксел структурного элемента совпадет с единичным пикселом бинарного изображения, то выполняется логическое сложение центрального пиксела структурного элемента с соответствующим пикселом выходного изображения;
* void ClosingFilter() – метод реализующий морфологическую операцию закрытия бинаризированного изображения. Операция к изображению применяет сначала операцию наращивания, что помогает избавиться от малых дыр и щелей, но при этом происходит увеличение контура объекта. Чтобы решить данную проблему производится эрозия, выполненная сразу после наращивания с тем же структурным элементом. Операция является основной;
* void DilatingFilter() – метод реализующий морфологическую операцию дилатации бинаризированного изображения. Метод основан на том, что структурный элемент применяется ко всем пикселам бинарного изображения. Каждый раз, когда начало координат структурного элемента совмещается с единичным бинарным пикселом, ко всему структурному элементу применяется перенос и последующее логическое сложение с соответствующими пикселами бинарного изображения;
* void OpeningFilter() – метод реализующий морфологическую операцию открытия бинаризированного изображения. Операция реализуется так: после операции эрозии применить операцию наращивания с тем же структурным элементом;
* bool HaveBlackNeighbors(Mat srcImg, int x, int y) – метод принимает исходное изображение и координаты пикселя в изображении, производит проверку имеет ли данный пиксель соседние черные пиксели;
* Scalar getColor(int cluster) – метод принимает номер кластера и возвращает цвет в который он окрашен;
* void showClusters(Mat srcImg) – метод принимает изображение и выводит уже результат кластеризации на экран;
* void AllocateObjects() – метод производит выделение и раскраску в цвета найденных объектов на изображении, данная операция разбита на 2 этапа. Первый – выделение контуров объектов, второй – закрашиваение внутреннего пространства каждого контура. Каждому из найденных объектов присваивается уникальный цвет. Диаграмма последовательностей метода изображена на рисунке 3.2;
* OpeningFilterBinary() – метод реализующий морфологическую операцию открытия бинаризированного изображения. Операция реализуется так: после операции эрозии применить операцию наращивания с тем же структурным элементом;
* void DilatingFilterManual() – метод реализующий морфологическую операцию дилатации бинаризированного изображения вручную. Метод основан на том, что структурный элемент применяется ко всем пикселам бинарного изображения. Каждый раз, когда начало координат структурного элемента совмещается с единичным бинарным пикселом, ко всему структурному элементу применяется перенос и последующее логическое сложение с соответствующими пикселами бинарного изображения;
* OpeningFilterBinaryManual() – метод реализующий морфологическую операцию открытия бинаризированного изображения вручную. Операция реализуется так: после операции эрозии применить операцию наращивания с тем же структурным элементом;
* void DenyFilterTwice() – данный метод производит отмену последнего морфологического преобразования или цифрового фильтра на два шага назад;

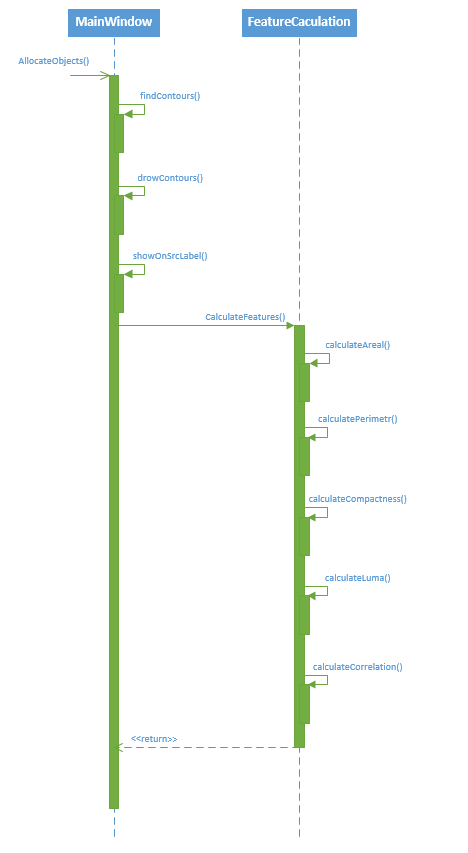


Рисунок 3.2 – Диаграмма последовательности метода

AllocateObjects()

* void StartClassification() – метод производит запуск обучения классификатора на основе предоставленных пользователем данных и непосредственно самой классификации. Диаграмма последовательностей метода изображена на рисунке 3.3;

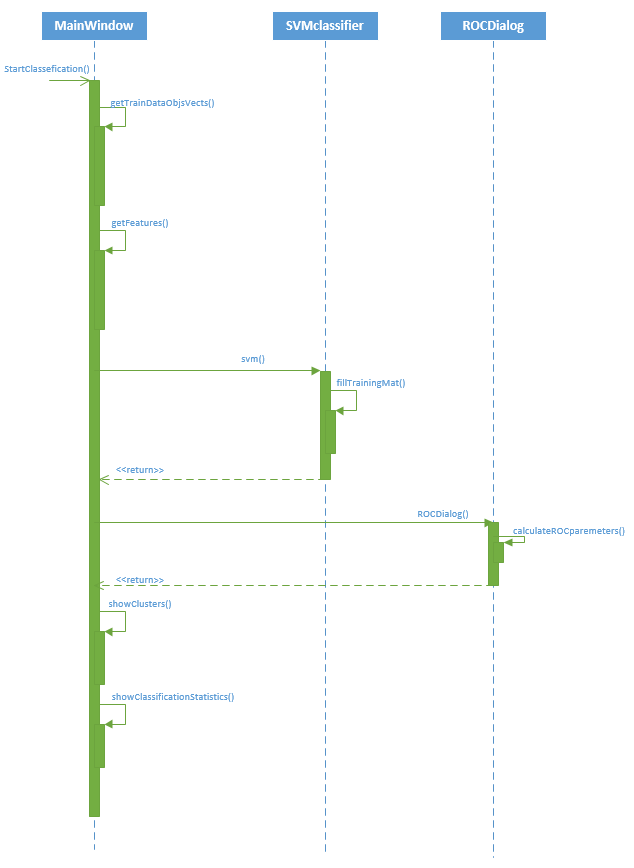


Рисунок 3.3 – Диаграмма последовательности метода

StartClassification()

* ColorSystem getLastColorSystem() const – метод возвращает значение поля LastColorSystem объекта, возвращаемое значение перечисляемого типа ColorSystem, метод предназначен для того чтобы получать доступ к полю LastColorSystem, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setLastColorSystem(const ColorSystem &value) – метод принимает значение поля LastColorSystem объекта, принимаемое значение перечисляемого типа ColorSystem, метод предназначен для того чтобы получать доступ к полю LastColorSystem, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int getCurrentIndex() const – метод возвращает значение поля CurrentIndex объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю CurrentIndex, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setCurrentIndex(int value) – метод принимает значение поля CurrentIndex объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю CurrentIndex, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* Mat getMatsrc() const – метод возвращает значение поля Matsrc объекта, возвращаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Matsrc, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setMatsrc(const Mat &value) – метод принимает значение поля Matsrc объекта, принимаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Matsrc, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* Mat getDuplicateMatSrc() const – метод возвращает значение поля DuplicateMatSrc объекта, возвращаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю DuplicateMatSrc, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setDuplicateMatSrc(const Mat &value) – метод принимает значение поля DuplicateMatSrc объекта, принимаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю DuplicateMatSrc, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* Mat getFirstImage() const – метод возвращает значение поля FirstImage объекта, возвращаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю FirstImage, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setFirstImage(const Mat &value) – метод принимает значение поля FirstImage объекта, принимаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю FirstImage, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QStack<Mat> getImgStack() const – метод возвращает значение поля ImgStack объекта, возвращаемое значение типа QStack<Mat>, метод предназначен для того чтобы получать доступ к полю ImgStack, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setImgStack(const QStack<Mat> &value) – метод принимает значение поля ImgStack объекта, принимаемое значение типа QStack<Mat>, метод предназначен для того чтобы получать доступ к полю ImgStack, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QStack<Mat> getImgBinaryStack() const – метод возвращает значение поля ImgBinaryStack объекта, возвращаемое значение типа QStack<Mat>, метод предназначен для того чтобы получать доступ к полю ImgBinaryStack, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setImgBinaryStack(const QStack<Mat> &value) – метод принимает значение поля ImgBinaryStack объекта, принимаемое значение типа QStack<Mat>, метод предназначен для того чтобы получать доступ к полю ImgBinaryStack, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QVector<Seed> getSeedVectorOldVertion() const – метод возвращает значение поля SeedVectorOldVertion объекта, возвращаемое значение типа QVector<Seed>, метод предназначен для того чтобы получать доступ к полю SeedVectorOldVertion, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setSeedVectorOldVertion(const QVector<Seed> &value) – метод принимает значение поля SeedVectorOldVertion объекта, принимаемое значение типа QStack<Mat>, метод предназначен для того чтобы получать доступ к полю SeedVectorOldVertion, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* bool getBinarizated() const – метод возвращает значение поля Binarizated объекта, возвращаемое значение типа bool, метод предназначен для того чтобы получать доступ к полю Binarizated, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setBinarizated(bool value) – метод принимает значение поля Binarizated объекта, принимаемое значение типа bool, метод предназначен для того чтобы получать доступ к полю Binarizated, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QString getFileName() const – метод возвращает значение поля FileName объекта, возвращаемое значение типа QString, метод предназначен для того чтобы получать доступ к полю FileName, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setFileName(const QString &value) – метод принимает значение поля FileName объекта, принимаемое значение типа QString, метод предназначен для того чтобы получать доступ к полю FileName, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* vector<Mat> getCSvector() const – метод возвращает значение поля CSvector объекта, возвращаемое значение типа vector<Mat>, метод предназначен для того чтобы получать доступ к полю CSvector, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setCSvector(const vector<Mat> &value) – метод принимает значение поля CSvector объекта, принимаемое значение типа vector<Mat>, метод предназначен для того чтобы получать доступ к полю CSvector, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* vector<Mat> getCSvectorColored() const – метод возвращает значение поля CSvectorColored объекта, возвращаемое значение типа vector<Mat>, метод предназначен для того чтобы получать доступ к полю CSvectorColored, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setCSvectorColored(const vector<Mat> &value) – метод принимает значение поля CSvectorColored объекта, принимаемое значение типа vector<Mat>, метод предназначен для того чтобы получать доступ к полю CSvectorColored, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция, является важным фактором хорошей архитектуры.

**3.3** Класс FeaturesCalculation

Класс обеспечивает подсчет выбранных в диалоговом окне параметров. Параметры могут могут быть геометрические, текстурные и яркостные. Геометрические рассчитываются на основе количества пикселей фигуры. Текстурные – на основе GLCM матрицы, которую строят по значениям самих пикселей. Яркостные – на основе значений пикселей.

Поля:

* srcImg – поле сохраняющее в себя актуальную версию изображения, является полем типа Mat;
* firstImg – поле хранящее первоначальное изображение, является полем типа Mat;
* seedVect – поле содержащее все объекты для которых считаются признаки, является полем типом QVector<Seed>;

Методы:

* void calculateArea() – метод вычисляющий площадь всех объектов на изображении, площадь вычисляется путем прохода по всему изображению и подсчета пикселей определенного цвета;
* void calculatePerimetr() – метод вычисляющий периметр всех найденных объектов;
* void calculateCompactness() – метод вычисляющий компактность всех объектов на изображении. Компактность вычисляется на основе уже посчитанного периметра и площади, поэтому для того чтобы избежать возможных ошибок все необходимые данные для этого методы должны быть вычислены заранее;
* bool HaveBlackNeighbors(int x, int y) – метод принимает координаты пикселя в исходном изображении, производит проверку имеет ли данный пиксель соседние черные пиксели и возвращает булево значение, которое является результатом проверки;
* QVector<Seed> GetSeedVector() – метод возвращающий вектор с объектами;
* void calculateLumaParameter(Mat srcImage) – метод принимает изображение и считает яркостную характеристику для каждого объекта, яркостной характеристикой является значение пикселя;
* void createGLCM(int indexOfSeed) – метод генерирует GLCM матрицу для каждого объекта для подсчета текстурных признаков, построение матрицы сводится к подсчету пар, рядом стоящих пикселей, у которых одинаковая яркость пар пикселей;
* void calculateContrast() – метод подсчитывает контраст для каждого объекта, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateHomogeneity() – метод считает гомогенность для всех объектов, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateDissimilarity() – метод считает различия для всех объектов на изображении, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateEnergy() – метод считает энергию для всех объектов на изображении, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateEntropy() – метод считает энтропию для всех объектов на изображении, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateCorrelation() – метод считает корреляцию для всех объектов на изображении;
* void calculateUandThigma(int index, float &U, float &thigmaSqr) – принимает индекс объекта, ссылки на U и thigma, и является вспомогательным для подсчета корреляции, данный параметр вычисляется на основании значений, которые хранятся в GLCM матрице;
* void calculateMatExpectation() – метод считает математическое ожидание для всех объектов на изображении, алгоритм использует разбиение изображения на уровни в зависимости от значения в пикселе;
* void calculateDispertion() – метод считает дисперсию для всех объектов на изображении;
* void calculateMassCenter() – метод считает центр масс для всех объектов на изображении, чтобы избежать возможных ошибок требуется вычислить заранее площадь объекта;
* void calculateElongation() – метод считает удлиненность для всех объектов на изображении, в этом методе вычисляются три момента, на основе которых в дальнейшем будет вычисляться сама удлиненность;
* void calculateSomeGeometryParam(PARAMETR) – метод принимает тип перечисление, который описывает ту операцию, которую необходимо выполнить. Диаграмма последовательностей метода изображена на рисунке 3.4;
* Mat getTrainingMat() const – метод возвращает значение поля Training\_mat объекта, возвращаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Training\_mat, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setTraining\_mat(const Mat &value) – метод принимает значение поля Training\_mat объекта, принимаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Training\_mat, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;

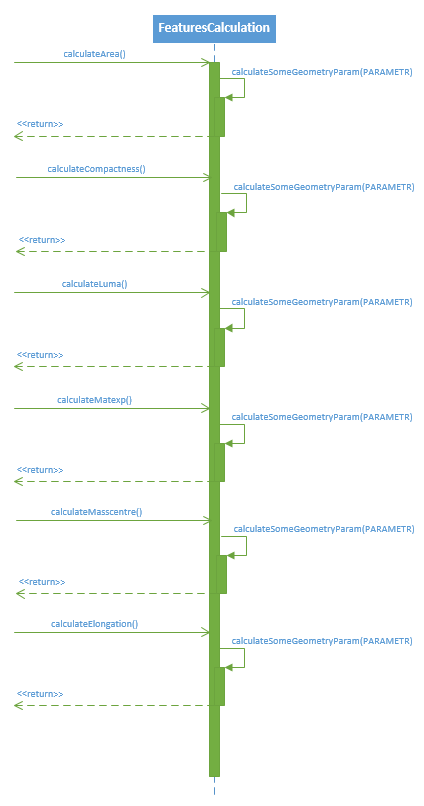


Рисунок 3.4 – Диаграмма последовательности метода

calculateSomeGeometryParam()

* void calculateTextureParameter(PARAMETR) – метод принимает тип перечисление, который описывает ту операцию, которую необходимо выполнить. Диаграмма последовательностей метода изображена на рисунке 3.5;

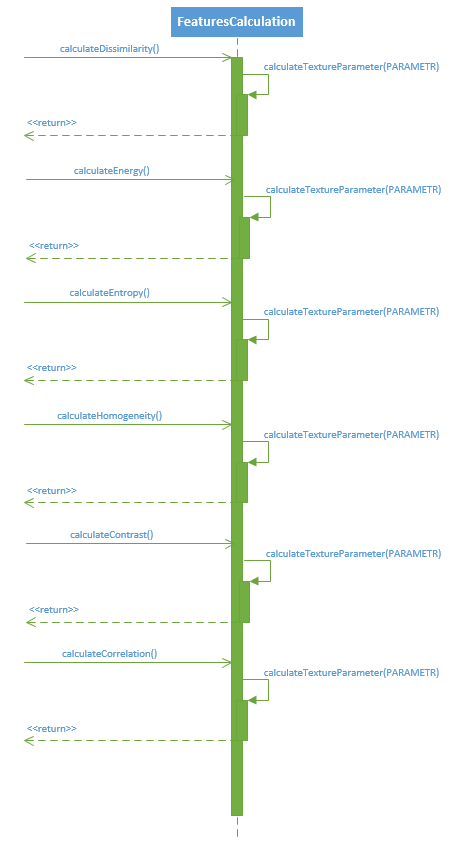


Рисунок 3.5 – Диаграмма последовательности метода

calculateTextureParameter()

**3.4** Класс SVMclassifier

Класс реализует заключительную задачу программного средства. Реализована SVM на основе открытой библиотеки OpenCV. Данные для обучения классификатора предоставляются пользователем путем выбора требуемых объектов в специальном диалоговом окне.

Поля:

* seedVect – поле хранящее все объекты, которые будут классифицироваться, поле типа QVector<Seed>;
* training\_mat – поле содержащее данные для обучения классификатора, поле типа Mat;
* featVect – поле хранящее признаки на основе которых будет происходить классификация объектов;

Методы:

* SVMclassifier(QVector<Seed> seedVector, QVector<int> featVector, int clusters, QVector<QVector<int>> trainDataObj) – переопределенный конструктор, используется для начальной инициализации класса данными;
* float\*\* CalculateTrainingData() – метод обучает классификатор на основе предоставленных данных;
* void fillObject(float \*arr, int numberOfSeed) – метод принимает массив объектов и номер объекта, классифицирует объекты, предоставленные ему;
* QVector<Seed> GetSeedVector() – метод возвращает все объекты с заполненными полями в данном классе;
* void FillTrainingMat() – метод заполняющий матрицу, на основе которой будет обучаться SVM. Диаграмма последовательностей метода изображена на рисунке 3.6;
* Mat getTraining\_mat() const – метод возвращает значение поля Training\_mat объекта, возвращаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Training\_mat, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setTraining\_mat(const Mat &value) – метод принимает значение поля Training\_mat объекта, принимаемое значение типа Mat, метод предназначен для того чтобы получать доступ к полю Training\_mat, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void fillObjectArray(float \*arr, int numberOfSeed) – метод принимает массив объектов и номер объекта, классифицирует объекты, предоставленные ему. Результат записывается в динамический массив \*arr.

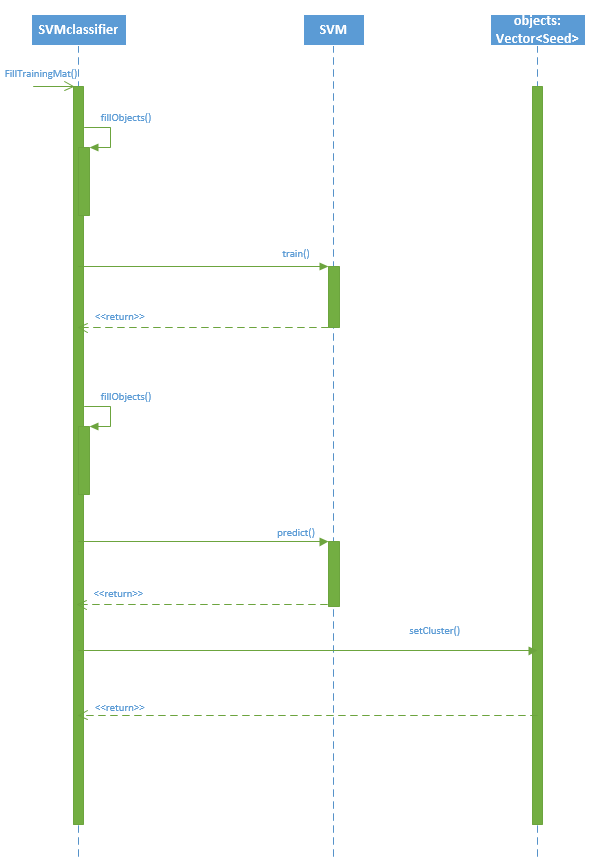


Рисунок 3.6 – Диаграмма последовательности метода

FillTrainingMat()

**3.5** Класс Seed

Этот класс реализует структуру зерна со значениями его параметров на основании которых будет принимать решение об отнесении его к определенном классу.

Поля:

* GLCM – поле хранящее GLCM матрицу, для подсчетов текстурных параметров, поле типа Mat;
* countOfPairs – поле содержащее количество пар, является полем типа int;
* contrast – значение контраста для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* homogeneity – значение гомогенности для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* dissimilarity – значение различие для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* entropy – значение энтропии для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* energy – значение энергии для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* correlation – значение корреляции для данного зерна, является полем типа float. Данный параметр является текстурным признаком и вычисляется на основе GLCM матрицы;
* countOfPixelsOnLevel – поле содержащее количество пикселей, является указателем на тип int;
* matExpect – значение математического ожидания для данного зерна, является полем типа float. Данный параметр является яркостным признаком и вычисляется на основе GLCM матрицы;
* dispersion – значение дисперсии для данного зерна, является полем типа float. Данный параметр является яркостным признаком и вычисляется на основе GLCM матрицы;
* centerMass – поле содержит точку центра масс для данного зерна, является полем типа QPoint. Данный параметр является геометрическим признаком и вычисляется на основе GLCM матрицы;
* correlationOfSeed – значение корреляции для конкретного объекта зерна, является полем типа float.
* elongation – значение удлиненности для данного зерна, является полем типа float. Данный параметр является геометрическим признаком и вычисляется на основе GLCM матрицы;
* probability – значение вероятности для данного зерна, является полем типа float;
* color – значение цвета для данного зерна после раскраски, является полем типа Scalar;
* area – значение площади для данного зерна, является полем типа int. Данный параметр является геометрическим признаком и вычисляется на основе GLCM матрицы;
* perimetr – значение периметра для данного зерна, является полем типа int. Данный параметр является геометрическим признаком и вычисляется на основе GLCM матрицы;
* compactness – значение компактности для данного зерна, является полем типа double. Данный параметр является геометрическим признаком и вычисляется на основе GLCM матрицы;
* cluster – поле показывает к какому классу отнесено зерно, является полем типа int;
* countOfFeatures – значение компактности для данного зерна, является полем типа double;
* luma – значение яркости для данного зерна, является полем типа int. Данный параметр является яркостным признаком и вычисляется на основе GLCM матрицы;
* countOfPixels – поле показывает сколько пикселей принадлежит данному зерну, является полем типа int;

Методы:

* void SetArea(int area) – метод принимает значение площади, параметр целочисленного типа int, метод предназначен для того чтобы получать доступ к полю Area, соблюдая принцип объектно ориентированного программирования – инкапсуляция;
* int GetArea() – метод возвращает значение площади, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю Area, соблюдая принцип объектно ориентированного программирования – инкапсуляция;
* void SetColor(Scalar color) – метод принимает значение цвета объекта, принимаемое значение типа Scalar, метод предназначен для того чтобы получать доступ к полю color, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* Scalar GetColor() – метод возвращает значение цвета объекта, возвращаемое значение типа Scalar, метод предназначен для того чтобы получать доступ к полю color, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetPerimetr(int perim) – метод принимает значение периметра объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю perimetr, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int GetPerimetr() – метод возвращает значение поля perimetr объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю perimetr, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetCompactness(double comp) – метод принимает значение поля compactness объекта, принимаемое значение числа с плавающей точкой типа double, метод предназначен для того чтобы получать доступ к полю compactness, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* double GetCompactness() – метод возвращает значение поля compactness объекта, возвращаемое значение числа с плавающей точкой типа double, метод предназначен для того чтобы получать доступ к полю compactness, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetCluster(int clust) – метод принимает значение поля cluster объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю cluster, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int GetCluster() – метод возвращает значение поля cluster объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю cluster, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetCountOfFeatures(int countOfFeatures) – метод принимает значение поля CountOfFeatures объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю CountOfFeatures, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int GetCountOfFeatures() – метод возвращает значение поля CountOfFeatures объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю CountOfFeatures, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetLuma(int luma) – метод принимает значение поля luma объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю luma, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int GetLuma() – метод возвращает значение поля luma объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю luma, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void SetCountOfPixels(int countOfPixels) – метод принимает значение поля countOfPixels объекта, принимаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю countOfPixels, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* int GetCountOfPixels() – метод возвращает значение поля countOfPixels объекта, возвращаемое значение целочисленного типа int, метод предназначен для того чтобы получать доступ к полю countOfPixels, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* Seed() – переопределенный конструктор, используется для начальной инициализации класса данными;
* ~Seed() – метод, является деструктором и служит для освобождения памяти и корректного уничтожения объекта;

**3.6** Класс ROCDialog

Класс реализует алгоритм построения и отображения ROC-кривой, кривая строится на основании предоставленных пользователем правильных данных о принадлежности зерен к классам и информации предоставляемой классификатором.

Поля:

* TP – значение классификации true positive для данных результатов классификации, является полем типа QVector<double>;
* FP – значение классификации false positive для данных результатов классификации, является полем типа QVector<double>;
* Data – поле хранит значение классификации для данных результатов классификации, является полем типа QVector<double>;
* seedVector – поле хранит все объекты зерен, является полем типа QVector<Seed>;
* fileName – поле хранит название открытого файла для того чтобы найти по нему правильные данные о классификации предоставленные пользователем, является полем типа QString;
* a1 – поле хранит координаты всех точек для кривой первого класса, является полем типа QVector<QPair<float, float> >;
* a2 – поле хранит координаты всех точек для кривой второго класса, является полем типа QVector<QPair<float, float> >;

Методы:

* double calcAUC(int labels, double scores, int n, int posclass) – метод принимает вероятности принадлежности объектов к классам, количество зерен, для какого класса будет вычисляться ROC-кривая, вычисляет все координаты точек ROC-кривой;
* void ROCforCluster(int \*labels) – метод вычисляет значения координаты точек на основе предоставленных данных;
* void smoothing(QVector<QPair<float, float> > &a) – метод принимает координаты точек и производит сглаживание кривой для лучшего визуального представления;
* double trapezoidArea(double X1, double X2, double Y1, double Y2) –метод принимает координаты точек и высчитает на их основе площадь под графиком ROC-кривой;
* void drawRocCurve(int posclass) – метод принимает номер кластера для которого требуется построить ROC-кривую, строит кривую;
* void calculateROCparemeters() – метод не принимает никаких параметров и высчитывает параметры требуемые для успешного построения ROC-кривой в дальнейшем;
* void makePlot() – метод производит графическое отображение самого окна на котором будут представлены результаты вычислений координат ROC-кривой;
* explicit ROCDialog(QString fileName, QVector<Seed> seedVect, QWidget \*parent = 0) – метод является конструктором данного окна;
* QVector<double> getFP() const – метод возвращает значение поля FP объекта, возвращаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю FP, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция, что является важным фактором правильной архитектуры;
* void setFP(const QVector<double> &value) – метод принимает значение поля FP объекта, принимаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю FP, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QVector<double> getTP2() const – метод возвращает значение поля TP2 объекта, возвращаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю TP2, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setTP2(const QVector<double> &value) – метод принимает значение поля TP2 объекта, принимаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю TP2, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* QVector<double> getFP2() const – метод возвращает значение поля FP2 объекта, возвращаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю FP2, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;
* void setFP2(const QVector<double> &value) – метод принимает значение поля FP2 объекта, принимаемое значение типа QVector<double>, метод предназначен для того чтобы получать доступ к полю FP2, соблюдая один из принципов объектно ориентированного программирования – инкапсуляция;

**3.7** Класс TrainingDataDialog

Данный класс реализует возможность выбора семян способом визуального отбора для предоставления классификатору эталонных признаков семян, которые относятся к данному класс. У пользователя есть возможность самому выбрать на изображении те объекты, которые по его мнению относятся к данному классу и видеть уже отобранные. Выделение отобранных реализуется путем добавления контура вокруг объекта. Для каждого класс контур имеет свой цвет, что помогает его отличить от других.

Поля:

* countOfClusters – поле содержит количество кластеров, которые предстоит выделить. Является полем целочисленным типом int;
* countOfObjsInCluster – поле содержит количество объектов уже добавленных к эталонной выборке для кластера, для которого выбираются объекты на текущий момент. Является полем целочисленным типом int;
* countOfObjsInGrBox – поле содержит количество объектов, которые сейчас отображены на интерфейсе для определенного кластера. Является полем целочисленным типом int;
* countActGrBox – поле содержит количество активных объектов;
* countOfThirds – поле содержит количество групп в которых уже достаточное количество объектов для нормальной эталонной выборки, для данного проекта было выбрано достаточное количество значение трех. Является полем целочисленным типом int;
* srcImg – поле содержит исходное изображение, на основании которого будет производиться выборка эталонных значение для последующей классификации. Является полем типа Mat;
* allocObjMat – поле содержит изображение с уже выделенными на данный момент объектами, на основании которого будет производиться выборка эталонных значение для последующей классификации. Является полем типа Mat;
* trainDataObjsVectrs – поле содержит массив эталонных объектов, который будет в последствии передан для дальнейшей тренировки машины опорных векторов. Является полем QVector<QVector<int> >;
* currentCluster – поле содержит номер кластера, объекты которого в данный момент выделяются. Является полем целочисленным типом int;
* seedVect – поле содержит массив всех семян, которые были найдены на данном изображении. Является полем целочисленным типом QVector<Seed>;

Методы:

* explicit TrainingDataDialog(QVector<Seed> seedVector, int countOfClusters, Mat allocObjMat, Mat srcImg, QWidget \*parent = 0) – метод принимает массив всех объектов, которые были найдены на изображении, количество кластеров на которое будут разнесены объекты, изображение для сохранения выделений и исходное изображение. Данный метод является конструктором и предназначается для начальной инициализации данного класса.
* ~TrainingDataDialog() – метод является деструктором и предназначается для освобождения ресурсов занимаемых классом.
* void colorOfСircuit(QPoint& pos) – метод принимает координаты точки в которой в данный момент было произведено нажатие курсора, предназначен определения цвета объекта который находится в данной координате;
* void fillLabels(int numberOfObject) – метод принимает номер выделенного объекта и предназначен для того чтобы изменить значение элемента пользовательского интерфейса соответствующего принятому номеру;
* QVector<QVector<int> > getTrainDataObjsVectrs() const – метод предназначен для возвращения массива объектов из класса, для дальнейшего использования;
* void showOnSrcLabel(Mat matImage) – метод принимает матрицу изображения и отображает его на пользовательском интерфейсе;
* Scalar getColor(int cluster) – метод принимает номер кластера и предназначен для получения цвета закрепленного за ним, метод возвращает найденный предназначенный для него цвет;
* bool HaveBlackNeighbors( int x, int y) – метод принимает координаты пикселя в изображении, производит проверку имеет ли данный пиксель соседние черные пиксели и возвращает булево значение результата проверки;
* void contourDetection(Scalar sc) – метод принимает цвет точки на которую был сделан клик курсора, предназначен для выделения контура объекта, который содержит данную точку;
* void showActiveGroupBox() – метод предназначен для отображения активных и доступных элементов пользовательского интерфейса;
* bool isEnoughObjForCluster(int cluster) – метод принимает номер кластера и предназначен для того чтобы проверять достаточно ли для него уже было выбрано эталонных объектов;
* void setCheckForCheckBox(int cluster, bool state) – метод принимает номер кластера и состояние в которое нужно установить элемент пользовательского интерфейса в соответствии с принятым номером;
* bool isAllCheckBoxAreChecked() – метод производит проверку на то, является ли все элементы пользовательского интерфейса в активном состоянии;

**3.8** Класс matDisplay

Данный класс предназначен для корректного выведения изображения, которое хранится в переменной типа Mat, на главное окно программного средства, на элемент, и определения координат нажатия мышью.

Методы:

* matDisplay(QWidget \*parent = 0) – конструктор по умолчания предназначен для начальной инициализации объектов данного класс. Принимает указатель на объект класса предка parent типа QWidget, устанавливает ему значение по умолчанию.
* ~matDisplay() – деструктор по умолчанию, при завершении использования объекта, вызывается и уничтожает объект и освобождает выделенную под него память. Аргументов не принимает.
* void mousePressEvent(QMouseEvent \*mouse\_event) – метод предназначен для определения момента нажатия мышью на элементе ImageLabelForm на окне выбора обучающих данных для классификатора. Так же метод определяет координаты нажатия мышью. Данные координаты необходимы для того, чтобы определить попал ли курсор мыши в контур какого либо объекта на изображении. Метод принимает аргумент – указатель на объект mouse\_event типа QMouseEvent, данный класс обрабатывает нажатия курсора и возвращает данные о нем.
* void sendMousePosition(QPoint& ) – метод является сигналом, он посылается другим классам при нажатии мышью на экране. Принимает ссылку на объект класса точки QPoint.

**3.9** Класс ImageLabelForm

Класс предназначен для корректного отображения изображения на всех окнах программного средства. Класс наследуется от класс предоставляемого Qt – QLabel.

Методы:

* explicit ImageLabelForm(QWidget \*parent = 0) – конструктор по умолчания предназначен для начальной инициализации объектов данного класс. Принимает указатель на объект класса предка parent типа QWidget, устанавливает ему значение по умолчанию.
* ~ImageLabelForm() – деструктор по умолчанию, при завершении использования объекта, вызывается и уничтожает объект и освобождает выделенную под него память. Аргументов не принимает.
* void setImage(Mat image) – метод предназначен для установки изображения в элемент пользовательского интерфейса. Принимает аргумент изображения image типа Mat.

**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программного модуля оценки качества зерна были реализованы некоторые нестандартные алгоритмы, такие как построение ROC-кривой, классификация, изменение цветовой системы.

**4.1** Построение ROC-кривой

Для представления пользователю качества классификации, производится построение ROC-кривой. Кривая строится на основе предоставленных пользователем идеальных данных классификатора и результатов полученных при работе классификатора. Производится перезапись данных из статического массива с идеальными данными в динамический.

for(int i = 0; i < seedVector.length(); i++)

{

idealClusterData[i] = idealClusterDataStatic[i];

}

В дальнейшем, расстояние от разделяющей классы прямой преобразуется в вероятность принадлежности объекта к одному из классов. Эти данные переписываются из вектора содержащего все объекты в динамический массив целочисленного типа int.

for(int j = 0; j < n; j++)

{

scores[j] = seedVector[j].probability;

scores2[j] = 1 - seedVector[j].probability;

}

Далее вычисляются параметры true positive и false positive для каждого объекта в массиве seedVector и сохраняются в массивы TPmas и FPmas. Данные точки высчитываются путем деления переменных TP и FP переменные P и N соответственно. P – количество правильных результатов классификации для класса, N – неправильных результатов.

if(fi != fprev)

{

fprev = fi;

FPprev = FP;

TPprev = TP;

this->TPmas.push\_back(TP/P);

this->FPmas.push\_back(FP/N);

}

if(label == posclass) {

TP = TP + 1;

}

else

{

FP = FP + 1;

}

this->TPmas.push\_back(TP/P);

this->FPmas.push\_back(FP/N);

После построения по полученным точкам графика ROC-кривой в большей части случаев необходимо производить процесс сглаживания. Данный процесс необходим для более удачного визуального представления кривой.

for(int i=0;i<a.size()-1;i++)

{

if(i+NO\_OF\_NEIGHBOURS+1<a.size())

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += a[i+j].first;

a[i].second += a[i+j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

} else

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += tmp[i-j].first;

a[i].second += tmp[i-j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

}

**4.2** Классификация

Разбиение всех найденных объектов на отдельные классы называется классификацией. За классификацию отвечает модуль принятия решений. Построен он на основе метода опорных векторов. Для начала процесса классификации методу требуются предоставленные пользователем данные. Данные содержат объекты с посчитанными характеристиками, которые пользователь предпочел выбрать как образец. Для каждого класса необходимо предоставить по три объекта относящихся к нему. Число три было выбрано для того чтобы увеличить шанс правильного отнесения объекта к классу.

Алгоритм классификации реализуется путем выполнения следующей последовательности шагов:

Шаг 1. Выбор обучающих данных:

1. Инициализация переменных. В методе используются следующие переменные:

* seedVect = QVector<Seed>(seedVector) – инициализирует массив с объектами семян из которых будет выбираться обучающая выборка;
* countOfClusters = countOfClusters – инициализируется количество кластеров для которых требуется заполнить обучающую выборку;

1. При нажатии мышью получаются координаты точки положения курсора;
2. Определяется к какому объекту на изображении относится координаты клика;
3. Закрашивается контур объекта цветом относящимся к выбираемому классу;

for(int i = 0; i < seedVect.length(); i++)

{

if(allocObjMat.at<Vec3b>(imgY, imgX)[0] == seedVect[i].GetColor().val[0] &&

allocObjMat.at<Vec3b>(imgY, imgX)[1] == seedVect[i].GetColor().val[1] &&

allocObjMat.at<Vec3b>(imgY, imgX)[2] == seedVect[i].GetColor().val[2])

{

if(isEnoughObjForCluster(currentCluster))

{

currentCluster++;

countOfObjsInCluster = 0;

}

countOfObjsInCluster++;

trainDataObjsVectrs[currentCluster].push\_back(i);

fillLabels(i);

contourDetection(seedVect[i].GetColor());

showOnSrcLabel(this->srcImg);

if(countOfObjsInCluster % countOfObjsInGrBox == 0)

{

setCheckForCheckBox(currentCluster,true);

currentCluster++;

countOfObjsInCluster = 0;

}

break;

}

1. Заносится выбранный объект в trainDataObjsVectrs и отобразить на интерфейсе, то, что объект добавлен.
2. Производится проверка на количество объектов для класса, если объектов достаточно, то ClusterGroupBox переводится в состояние enable и происходит переход к следующему классу.
3. После заполнения всех требуемых данных процесс завершается и диалог возвращает значение trainDataObjsVectrs;

Шаг 2. Обучение классификатора:

1. Инициализация переменных. В методе используются следующие переменные:

* seedVect = QVector<Seed>(seedVector) – инициализирует массив с объектами семян из которых будет выбираться обучающая выборка;
* featVect = QVector<int>(featVector) – инициализируется массив с признаками объекта, которые были выбраны ранее для того, чтобы быть условием классификации;
* trainigDataObjs = QVector<QVector<int> > (trainDataObj) – массив объектов выбранных для того, чтобы быть обучающей выборкой классификатора;

1. Производится подсчет количества переданных объектов в обучающей выборке trainigDataObjs;

for(int i = 0; i < trainigDataObjs.length(); i++)

{

for(int j = 0; j < trainigDataObjs[i].length(); j++)

{

countOfOb++;

}

}

1. Каждый объект представляется в виде динамического массива целочисленного типа int;

for(int i = 0; i < trainigDataObjs.length(); i++)

{

for(int j = 0; j < trainigDataObjs[i].length(); j++)

{

labels[indexOfObject++] = i;

fillObject(trainingData[k++], trainigDataObjs[i][j]);

}

}

1. Все преобразованные данные аккумулируются в trainingDataMat типа Mat, размерность матрицы задает количество объектов;

Mat trainingDataMat(countOfOb, featVect.length(), CV\_32FC1, trainingData);

1. Производится обучение классификатора путем вызова метода train у объекта SVM. В качестве параметра в метод передаются данные выбранные заранее пользователем – trainingDataMat;

CvSVMParams params;

params.svm\_type = CvSVM::C\_SVC;

params.kernel\_type = CvSVM::LINEAR;

params.term\_crit = cvTermCriteria(CV\_TERMCRIT\_ITER, 100, 1e-6);

CvSVM SVM;

SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);

1. Все объекты преобразуются в динамические массивы целочисленного типа int;
2. Основным шагом в классификации является вызов метода predict у объекта SVM. Метод производит классификацию и возвращает для каждого объекта значение расстояния от разделяющей классы прямой, по нему определяется принадлежность объекта к классу;
3. Заключительный этап классификации – это преобразование расстояния до разделяющей прямой от объекта в вероятность принадлежности объекта к классу;

float confidence;

for(int i = 0; i < seedVect.length(); i++)

{

fillObject(object, i);

Mat objectMat(1, featureLength, CV\_32FC1, object);

float cl2 = SVM.predict(objectMat, true);

confidence = 1.0 / (1.0 + exp(-cl2));

float cl = SVM.predict(objectMat);

seedVect[i].SetCluster(cl);

seedVect[i].probability = confidence;

}

**4.3** Изменение цветовой системы

Часто, для решения разного рода задач требуется различное представление данных. Получить цветовая система RGB не всегда предоставляет требуемые данные, в таких случаях возникает потребность в расширении или изменении цветовых каналов изображения. В дипломной работе предоставлена возможность использования четырех дополнительных видов цветовых систем: HLS, HSV, CMYK и Lab.

Рассмотрим алгоритм функции преобразующей цветовую систему RGB в цветовую систему CMYK содержит следующие пункты:

1. Инициализация данных в которые будут сохранены преобразованные каналы;

Mat channel[4];

for (int i = 0; i < 4; i++)

{

cmyk.push\_back(cv::Mat(img.size(), CV\_8UC1));

channel[i] = Mat::eye(img.size(), CV\_8UC1);

}

1. Разделение изображения в цветовой системе RGB на отдельные, составляющие ее, каналы: Red, Green, Blue.

std::vector<cv::Mat> rgb;

cv::split(img, rgb);

1. Производится перекодирование каналов RGB в каналы CMYK. Требуется произвести перебор всех пикселей изображения и, используя весовые коэффициенты, вычислить новые значения пикселей.

for (int i = 0; i < img.rows; i++)

{

for (int j = 0; j < img.cols; j++)

{

float r = (int)rgb[2].at<uchar>(i, j) / 255.;

float g = (int)rgb[1].at<uchar>(i, j) / 255.;

float b = (int)rgb[0].at<uchar>(i, j) / 255.;

float k = std::min(std::min(1- r, 1- g), 1- b);

uchar ch = channel[0].at<uchar>(i, j);

channel[0].at<uchar>(i, j) = (1 - r - k) / (1 - k) \* 255.;

channel[1].at<uchar>(i, j) = (1 - g - k) / (1 - k) \* 255.;

channel[2].at<uchar>(i, j) = (1 - b - k) / (1 - k) \* 255.;

channel[3].at<uchar>(i, j) = k \* 255.;

}

}

1. Записать полученные каналы в выходной буфер, который передается в метод по указателю.

cmyk[0] = channel[0].clone();

cmyk[1] = channel[1].clone();

cmyk[2] = channel[2].clone();

cmyk[3] = channel[3].clone();

1. Для более удобного восприятия каналов пользователем, требуется придать каждому каналу свой цвет.

std::string cmyk\_labels[3] = {"[C]yan", "[M]agenta", "[Y]ellow"};

double std\_values[3][3] = {{255, 255, 0}, {255, 0, 255}, {0, 255, 255}};

cmykColored = showChannels(img, channel, cmyk\_labels, std\_values, COLOR\_HSV2BGR, true);

Преобразование в цветовую систему CMYK позволяет расширить каналы RGB.

Функция преобразующая RGB в цветовую систему HSV содержит следующие пункты:

1. Инициализация данных в которые будут сохранены преобразованные каналы;

Mat channel[3];

for (int i = 0; i < 3; i++)

{

hsvVector.push\_back(cv::Mat(img.size(), CV\_8UC1));

}

1. Вызов функции преобразования из цветовой системы RGB в систему HSV.

cvtColor(img,hsv,CV\_BGR2HSV);

1. Разделение преобразованного изображения в цветовой системе HSV на отдельные, составляющие ее, каналы: Hue, Saturation, Value.

split(hsv, channel);

1. Запись полученных разделенных каналов в выходной буфер, переданный в метод по указателю.

hsvVector[0] = channel[0].clone();

hsvVector[1] = channel[1].clone();

hsvVector[2] = channel[2].clone();

1. Преобразование каналов из полутона в цветное изображение для лучшего визуального восприятия путем добавления каждому каналу своего цвета.

std::string hsv\_labels[3] = {"[H]ue", "[S]aturation", "[V]alue"};

double std\_values[3][3] = {{1, 255, 255}, {179, 1, 255}, {179, 0, 1}};

hsvVectorColored = showChannels(img, channel, hsv\_labels, std\_values, COLOR\_HSV2BGR, false);

Рассмотрим алгоритм функции преобразующей цветовую RGB в цветовую систему HLS содержит следующие пункты:

1. Инициализация данных в которые будут сохранены преобразованные каналы:

* vector<Mat> hlsVectorColored = NULL – vector в который будут помещаться каналы уже преобразованный из полутона в цвет, инициализируется NULL;
* Mat hls = Mat() – изображение в цветовой системе HLS, инициализируется пустой матрицей;
* Mat channel[3] – статический массив для сохранения каналов;

for (int i = 0; i < 3; i++)

{

hlsVector.push\_back(cv::Mat(img.size(), CV\_8UC1));

}

1. Вызов функции преобразования из цветовой системы RGB в систему HLS.

cvtColor(img,hls,CV\_BGR2HLS);

1. Разделение преобразованного изображения в цветовой системе HLS на отдельные, составляющие ее, каналы: Hue, Lightness, Saturation.

split(hls, channel);

1. Запись полученных разделенных каналов в выходной буфер, переданный в метод по указателю.

hlsVector[0] = channel[0].clone();

hlsVector[1] = channel[1].clone();

hlsVector[2] = channel[2].clone();

1. Преобразование каналов из полутона в цветное изображение для лучшего визуального восприятия путем добавления каждому каналу своего цвета.

std::string hls\_labels[3] = {"[H]ue", "[L]ightness", "[S]aturation"};

double std\_values[3][3] = {{1, 100, 50}, {179, 1, 255}, {179, 69, 1}};

hlsVectorColored = showChannels(img, channel, hls\_labels, std\_values, COLOR\_HLS2BGR, false);

Рассмотрим алгоритм функции преобразующей цветовую RGB в цветовую систему Lab содержит следующие пункты:

1. Инициализация данных в которые будут сохранены преобразованные каналы:

* vector<Mat> labVectorColored = NULL – vector в который будут помещаться каналы уже преобразованный из полутона в цвет, инициализируется NULL;
* Mat lab = Mat() – изображение в цветовой системе Lab, инициализируется пустой матрицей;
* Mat channel[3] – статический массив для сохранения каналов;

for (int i = 0; i < 3; i++)

{

labVector.push\_back(Mat(img.size(), CV\_8UC1));

}

1. Вызов функции преобразования из цветовой системы RGB в систему HLS.

cvtColor(img,lab,CV\_BGR2Lab);

1. Разделение преобразованного изображения в цветовой системе Lab на отдельные, составляющие ее, каналы: Luminance, Adimension, Bdimention.

split(lab, channel);

1. Запись полученных разделенных каналов в выходной буфер, переданный в метод по указателю.

labVector[0] = channel[0].clone();

labVector[1] = channel[1].clone();

labVector[2] = channel[2].clone();

1. Преобразование каналов из полутона в цветное изображение для лучшего визуального восприятия путем добавления каждому каналу своего цвета.

**4.4** Построение GLCM матрицы

Для подсчета текстурных параметров объектов требуется построить GLCM матрицу. Данная матрица строится на основе значений яркости пикселей объекта.

Рассмотрим алгоритм функции формирующую GLCM матрицу, которая содержит следующие пункты:

1. Инициализация двух матриц из которых будет в дальнейшем формироваться сама основная матрица.

Mat GLCMtrasposed = Mat::zeros(32, 32, CV\_8UC1);

Mat GLCMsymmetric = Mat::zeros(32, 32, CV\_8UC1);

1. Заполнение матрицы GLCMtrasposed путем транспонирования исходного изображения объекта.

for(int i=0; i< seedVect[indexOfSeed].GLCM.rows; i++)

for(int j=0; j< seedVect[indexOfSeed].GLCM.cols; j++)

{

GLCMtrasposed.at<uchar>(j,i) = seedVect[indexOfSeed].GLCM.at<uchar>(i,j);

}

1. Формирование матрицы симметрии GLCMsymmetric путем складывания пикселей пустой матрицы с пикселями матрицы транспонированной.

for(int i=0; i<seedVect[indexOfSeed].GLCM.rows; i++)

for(int j=0; j<seedVect[indexOfSeed].GLCM.cols; j++)

{

GLCMsymmetric.at<uchar>(i,j) = (seedVect[indexOfSeed].GLCM.at<uchar>(i,j) + GLCMtrasposed.at<uchar>(i,j));

}

1. Запись матрицы симметрии в GLCM матрицу объекта.

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – процесс исследования программного обеспечения с целью получения информации о качестве продукта. Тестирование является одним из важных этапов разработки, поскольку при написании программного кода невозможно предусмотреть все случаи, и ошибки в работе программы неизбежны. Кроме того, возможно появление ошибок в случае внесения дополнительного функционала в программу даже в уже отлаженном рабочем коде.

Тестирование программы направлено на доказательства её соответствия заявленным требованиям. Таким образом, основной целью тестирования является выделение несоответствия функционирования программы заявленным требованиям и устранение их.

Тестирование программы проводилось в два этапа:

* поэтапное тестирование отдельно каждого модуля в процессе написания программного кода;
* полное тестирование программы после окончания процесса написания программного кода.

Оба этапа являются достаточно важными, ни один из них невозможно исключить. Например, без модульного тестирования, при анализе работы программы в целом, будет происходить достаточное количество сбоев, выявить и локализовать которые может оказаться достаточно сложным заданием, в то время как при анализе работы одного модуля неисправность оказывается достаточно очевидной. И обратный случай, работоспособность каждого компонента в отдельности не гарантирует корректное поведение всей программы в целом.

Тестирование программы проводилось на следующих компьютерах:

* Intel Core i5 2 ядра по 1,3 ГГц, оперативная память 4ГБ 1600МГц DDR3, видеокарта Intel HD Graphics 5000 1536МБ. Операционная система OS X El Capitan 10.11.4;
* Intel Core i5 2 ядра по 1,6 ГГц, оперативная память 4ГБ 1600МГц DDR3 , видеокарта Intel HD Graphics 6000 1536МБ. Операционная система OS X Yosemite 10.10.2;

В программе предусмотрено самотестирование модуля. При первом запуске происходит проверка, все ли файлы находятся на месте. Если какой-то файл не найден, предпринимается попытка восстановить его, если восстановление невозможно, в таком случае происходит информирование пользователя о том, что программа может работать некорректно и происходит игнорирование недостающего файла.

В программном средстве реализовано тестирование входных данных и тестирования корректности результатов проведенных операций.

Первоначально происходит проверка на правильность полученного пути к изображению. Если тест пройден, то происходит открытие изображения. При открытии графического файла необходимо удостоверится, что он существует и открыт корректно. Проверка происходит каждый раз, когда пользователь принимает решение об открытии нового изображения. Проверка на корректность необходима, потому что при неверном открытии графического файла программе будут предоставлены некорректные данные, что влечет за собой неправильную работу всех последующих алгоритмов в программном средстве. В случае не прохождения теста пользователю будет представлено диалоговое окно с сообщением об этом.

if(!nameOfOpenFile.isEmpty())

{

if(!matsrc.empty())

{

showOnSrcLabel(matsrc);

}

else

{

QMessageBox::information(this, tr("Loading image"), tr("Can't load %1.").arg(nameOfOpenFile));

}

}

Пример такого сообщения приведен на рисунке 5.1.

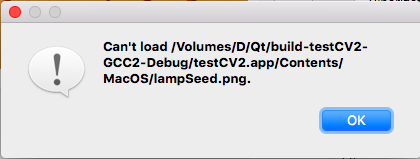


Рисунок 5.1 – Пример сообщения о невозможности открыть файл

В каждой операции, в которой алгоритму необходимо оперировать графическим файлом производится проверка на существование данного файла. Тестирование данной исключительной ситуации необходимо, потому что попытка прочитать или записать в изображение, которое не существует, приведет к преждевременному завершению программы.

if(!matsrc.empty())

{

//...

}

else

{

QMessageBox::information(this, tr("Loading image"), tr("Can't load %1.").arg(nameOfOpenFile));

}

В методах реализующих морфологические операции и цифровые фильтры, кроме, выше приведенной проверки, необходимо произвести тестирование переменной binarizated на положительное значение. Данная переменная устанавливается в true, в случае, если бинаризация изображения была уже произведена. Тестирование необходимо, потому что алгоритмам морфологических операций и цифровых фильтров необходимо изображение в котором присутствую только черный и белый цвета. Если тест не пройден, то пользователю будет продемонстрировано диалоговое окно с сообщением об этом.

if(binarizated == true)

{

//...

}

else

{

QMessageBox::information(this, tr("Dilating filter"), tr("Image is not binarizated"));

}

Пример такого сообщения приведен на рисунке 5.2.

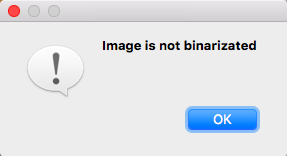


Рисунок 5.2 – Пример сообщения о требовании провести бинаризацию

При проведении классификации производится тестирование количества данных предоставляемых пользователем для обучения классификатора. Количество данных не должно быть меньше того числа, которое пользователь ввел с пользовательского интерфейса. Если тест не пройден, то пользователю будет продемонстрировано диалоговое окно с сообщением об этом. Пользователю программы будет необходимо повторить выбор обучающих данных. Данный тест необходим, так как при отсутствии данных для хотя бы из одного классов объектов, алгоритм классификации не будет произведен корректно.

if(trainingData.length() < ui->clusterSpinBox->value())

{

QMessageBox::information(this, tr("Classification"), tr("Need more training data, click on \"SVM\" button again"));

}

Пример такого сообщения приведен на рисунке 5.3.

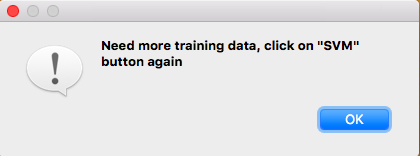


Рисунок 5.3 – Пример сообщения от классификатора

Для обнаружения остальных исключительных ситуаций проводилось функциональное тестирование. Тестирование уже готовой программы проводилось сразу несколькими пользователями на разных компьютерах с разной производительностью и разными версиями операционной системы. Тесты проводимые пользователями помогают окончательно удостовериться в надежности разработанного программного средства. После каждого не пройденного теста, был проведен тщательный анализ ситуации приведшей к сбою, после чего было проведено исправление критического кода вызвавшего ошибку.

Тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Окно программы | Содержание теста | Ожидаемый результат | Тест пройден |
| 1 | 2 | 3 | 4 |
| MainWindow | Реакция программы на нажатие всевозможных клавиш на клавиатуре. | На нажатие клавиш программа не реагирует. | да |
| MainWindow | Вызов окна выбора файла нажатием на кнопку «Open». | Показывается окно с выбора файла. | да |
| MainWindow | Нажатие кнопки «CMYK» | Изображение переводится в данную цветовую систему «CMYK», все слои отображаются на экране. | да |
| MainWindow | Нажатие кнопки «HSV». | Изображение переводится в данную цветовую систему «HSV», все слои отображаются на экране. | да |
| MainWindow | Нажатие кнопки «HLS». | Изображение переводится в данную цветовую систему «HLS», все слои отображаются на экране. | да |
| MainWindow | Нажатие кнопки «Lab». | Изображение переводится в данную цветовую систему «Lab», все слои отображаются на экране. | да |
| MainWindow | Нажатие кнопки «Deny». | Изображение возвращается в исходную цветовую систему – «RGB». | да |
| MainWindow | Нажатие кнопки «ToGrayScale». | Изображение преобразуется из цветного в полутоновое изображение, конечный результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Binarization» | Производится операция бинаризации, результат выводится на экран. | да |

*Продолжение таблицы 5.1*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| MainWindow | Нажатие кнопки «Otsu» в разделе «Binarization». | Производится операция бинаризации методом Оцу, результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Deny last binarization» в разделе «Binarization». | Выводит на экран исходное изображение для бинаризации. | да |
| MainWindow | Нажатие кнопки «Median» в разделе «Filters». | Производится фильтрацию изображения с помощью медианного фильтра, результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Erode» в разделе «Filters». | Производится морфологическую операцию эрозии, результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Dilating» в разделе «Filters». | Производится морфологическую операцию дилатации, результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Closing» в разделе «Filters». | Производится морфологическую операцию закрытия, результат выводится на экран. | да |
| MainWindow | Нажатие кнопки «Opening» в разделе «Filters». | Производится морфологическую операцию открытия, результат выводится на экран. | да |

Как видно из таблицы, приложение хорошо справилось с тестами, что говорит о высокой его работоспособности.

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске программы пользователь увидит окно, примерный вид которого изображен на рисунке 6.1.

**6.1** Окно SСlassifier

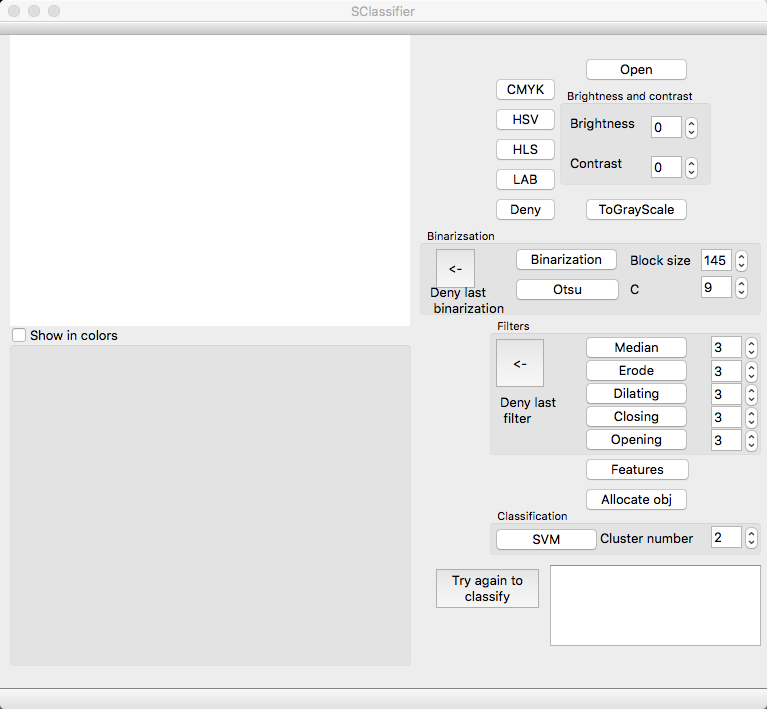


Рисунок 6.1 – Внешний вид главного окна

В левой части главного окна находятся две области, в которых будут отображаться результаты действий пользователя. Между ними располагается поле «Show in colors». Если в поле установлена галочка, то при смене цветовой системы со стандартной на любую другую из предоставленных каналы будут окрашены в цвета для лучшего визуального восприятия, иначе будут в полутоновом виде.

В правой части окна находятся кнопки управления программой.

Для начала работы требуется нажать кнопку «Open» и выбрать графический файл для анализа. Далее пользователь может путем нажатия кнопок «CMYK», «HSV», «HLS» и «Lab» выбрать нужную ему цветовую систему. Каналы системы будут отображены на нижней области. Кнопка «Deny» требуется, если пользователь решит отменить преобразование и вернуться к стандартной системе «RGB».

Справа от кнопок смены цветовой системы находятся два элемента счетчика:

* «Brightness» – уровень яркости;
* «Contrast» – уровень контраста;

Яркость управляется нажатием стрелок на элементе счетчика яркости. Нажатие на кнопку «Вверх» ведет к увеличению яркости изображение, нажатие «Вниз» – к уменьшению уровня яркости.

Контраст управляется нажатием стрелок на элементе счетчика констраста. Нажатие на кнопку «Вверх» ведет к увеличению контраста изображение, нажатие «Вниз» – к уменьшению уровня контраста.

Ниже располагается кнопка «ToGrayScale». Нажатие данной кнопки приводит к преобразованию изображение из цветного в полутоновое. Преобразование является необратимым, в случае, если пользователь решит изменить контраст или цветовую систему уже преобразованого изображения, ему придется загружать файл снова, путем нажатия на кнопку «Open».

Ниже находится область «Binarization».

Тут содержится три кнопки:

* «Binarization» – простая адаптивная бинаризация;
* «Otsu» – адаптивная бинаризация Оцу;
* «Deny last binarization» – отмена последней бинаризации;

Так же тут находятся два элемента счетчика:

* «Block size» – размер окна бинаризации;
* «C» – компонента С для простой адаптивной бинаризации;

С помощью кнопок «Binarization» и «Otsu» может производиться адаптивная бинаризация изображения. Предварительно изображение должно быть переведено в полутон, иначе будет исключительная ситуация и пользователь увидит диалоговое окно с сообщение о необходимости перевода изображения в полутон. С помощью кнопки «Deny last binarization» можно отменить последнюю бинаризацию, это значит, что операция бинаризации обратима. В элементах счетчиках «Block size» и «C» пользователь может задать аргументы для бинаризаций. От выбора значений данных элементов зависит результат бинаризации. Меняя их можно получить наилучший результат для конкретного изображения. Значения изменяются путем нажатия кнопок со стрелками «Вверх» и «Вниз».

Ниже области «Binarization» находится область «Filters».

Область содержит пять элементов счетчиков и шесть кнопок:

* «Median» – применить медианный фильтр;
* «Erode» – применить морфологическую операцию эрозии;
* «Dilating» – применить морфологическую операцию дилатации;
* «Closing» – применить морфологическую операцию закрытия;
* «Opening» – применить морфологическую операцию открытия;
* «Deny last filter» – отменить последнюю операцию;

Пользователь имеет возможность дообработать графический файл с помощью фильтров и морфологических операций. При наличии на изображении шума рекомендуется использовать медианный фильтр. Так же есть возможность увеличить или уменьшить объекты, данные операции применяются, когда нужно разъединить два объекта или нарастить объект, до определенных нужных пользователю объектов. Каждый результат будет выводиться на экран, в нижнюю графическую область. У каждой операции есть относящийся к ней элемент счетчика. Счетчиком задается размер ядра операции. В зависимости от размера ядра будет изменяться результат проведенной операции. С помощью кнопки «Deny last filter» можно отменить последнюю операцию, это значит, что операция фильтрации обратима.

Ниже находится кнопка «Features», нажатие на которую вызывает открытие окна «Features». Диалоговое окно «Features» представлено на рисунке 6.2.

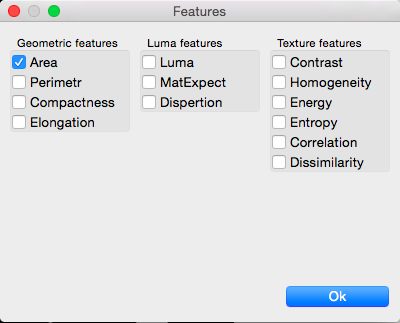


Рисунок 6.2 – Внешний вид окна выбора параметров

На диалоговом окне пользователю предоставляется возможность выбора параметров объектов, которые будут вычислены и впоследствии использоваться при классификации. Первоначально при открытии окна предустановлен выбор параметра Area. Это сделано для того, чтобы хотя бы один параметр был использован, так как классификатору требуется один и более параметров. Выбор различных параметров влияет на дальнейшую классификацию. При нажатии на кнопку «Оk», окно закрывается и управление передается в окно «Sclassifier».

Ниже находится кнопка «Allocate obj». Нажатие на эту кнопку производится выделение связных областей на изображении. Итогом операции является выделение каждого объекта уникальным цветом. Результат операции выводится на экран. Операция продемонстрирована на рисунке 6.3.

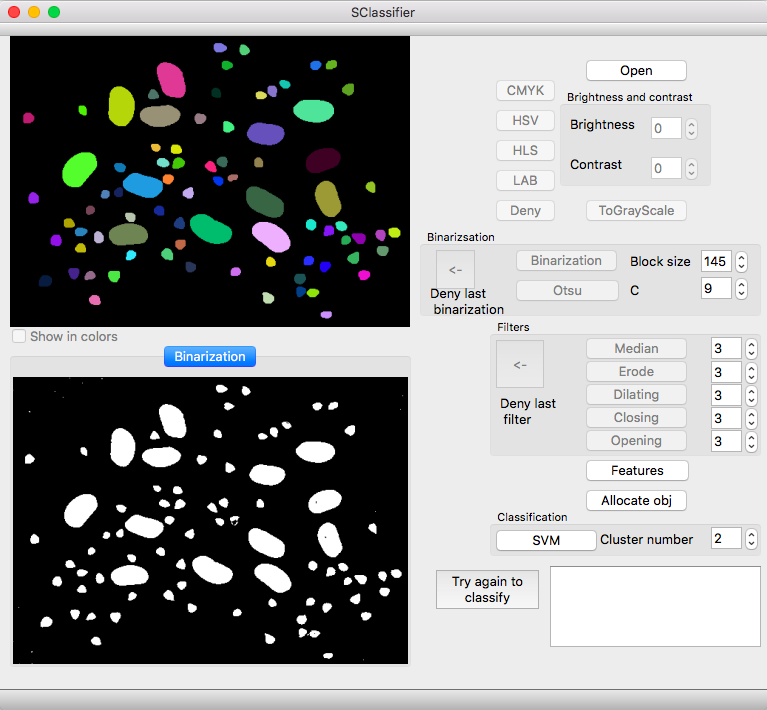


Рисунок 6.3 – Результат операции выделения связных областей

Еще ниже находится область «Classification». В нем содержится кнопка «SVM» и элемент счетчика. Нажатие на кнопку запускает процесс классификации включающий несколько шагов:

Шаг 1. Вызов окна «Training data choosing». Данное окно предназначено для выбора пользователем данных для обучения классификатора. Выбор происходит путем нажатия курсором на объект класса на изображении. Объекты для классов выбираются последовательно. Если объектов для класса менее чем три, то пользователь может установить флаг «Enough» в активное состояние и для класса выборка обучающих данных будет прекращена. Нажатие на кнопку «Deny» отменяет выборку данных и закрывает окно. Нажатие на «Ok» сохраняет данные и возвращает управление окну «Sclassifier». Окно «Training data choosing» представлено на рисунке 6.4.

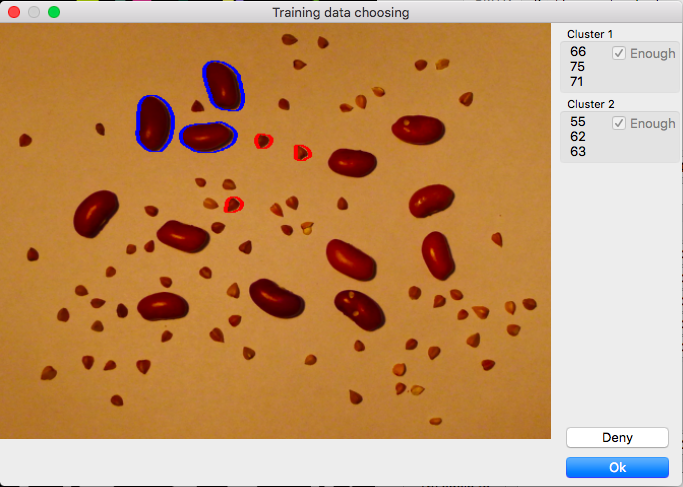


Рисунок 6.4 – Выбор обучающих данных

Шаг 2. Запускается процесс классификации на основе данных полученных в предыдущем шаге. Количество классов получается из элемента счетчика. Результаты выводятся на экран. Так же имеется кнопка «Try again to classify» нажатие на нее позволяет отменить результаты классификации и начать процесс сначала.

После завершения работы классификатора будет рассчитана и выведена на экран ROC-кривая. Она демонстрирует качество проведенной классификации. Кривая отображается в отдельном окне ROC-curve. Программное средство строит в одном окне сразу все графики всех классов. Для удобства визуального восприятия и их различия кривые раскрашены в разные цвета. Пример такого изображения представлен на рисунке 6.5.

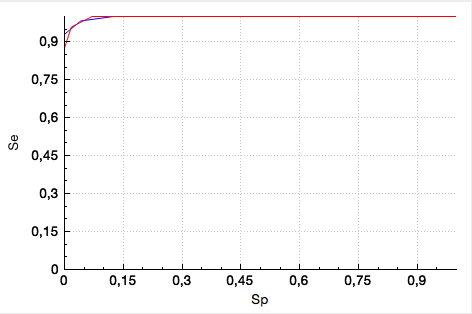


Рисунок 6.5 – Построение ROC-кривых

Для успешной работы программы ей потребуется процессор Intel Core i5 2 ядра по 1,3 ГГц, оперативная память 4ГБ 1600МГц DDR3, видеокарта Intel HD Graphics 5000 1536МБ. Операционная система OS X El Capitan 10.11.4.

**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

**7.1** Характеристика программного продукта

Целью данного дипломного проекта является разработка программного средства для анализа зерен сельскохозяйственных культур. Данный программный продукт служит для упрощения процесса анализа и классификации зерен.

В настоящее время программное обеспечение представляет собой товар, выпускаемый научно-техническими организациями, который поставляется покупателям по рыночным ценам. Экономический эффект зависит от объема затрат на разработку проекта, уровня цены на разработанный программный продукт и объема продаж. Выбор эффективных проектов ПО требует их экономической оценки и расчета экономического эффекта.

Целью технико-экономического обоснования является определение экономической выгодности создания рассматриваемого программного обеспечения и дальнейшего его применения.

**7.2** Экономический эффект у разработчика

Данный программный модуль относится к 1 категории сложности, так как он работает в реальном времени и может собирать информацию с нескольких удалённых объектов. При этом дополнительный коэффициент сложности равняется 1,12. По степени новизны ПС относится к категории А с коэффициентом новизны , так как принципиальных аналогов выявлено не было. Но данный проект не подразумевает в себе использование принципиально нового типа ЭВМ или ОС.

При разработке модуля используются существующие технологии и средства разработки, которые охватывают около 20 – 30% реализуемых функций, поэтому коэффициент использования стандартных модулей принимается равным 0,8.

**7.2.1** Определение объема и трудоемкости ПО

Для того, чтобы рассчитать плановую смету затрат на разработку ПО, требуется определить общий объем ПС (). Стоимостная оценка программного средства у разработчиков предполагает составление сметы затрат. Смета затрат – это полный расчет расходов предприятия на производство и реализацию продукции за определенный календарный период.

Объем разрабатываемых функций определяется по каталогу функций (см. таблицу 7.1).

Таблица 7.1 – Каталог функций ПО

|  |  |  |  |
| --- | --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функций по каталогу | Объем функций уточнённый |
| 1 | 2 | 3 | 4 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 | 420 |
| 107 | Организация ввода\вывода информации в интерактивном режиме | 150 | 130 |
| 202 | Формирование баз данных | 1020 | 850 |
| 207 | Манипулирование данными | 9550 | 9100 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 490 |
| 507 | Обеспечение интерфейса между компонентами | 970 | 1250 |
| 601 | Проведение тестовых испытаний прикладных программ в интерактивном режиме | 3200 | 2700 |
| 707 | Графический вывод результатов | 480 | 530 |
|  | Итого | 16230 | 15470 |

На основе общего объема ПС рассчитывается нормативная трудоемкость ПО (). Так как общий объем ПС = 15470, категория сложности ПО – 1, нормативная трудоемкость = 628. Данная система обеспечивает интерактивный доступ, телекоммуникационную обработку данных и управление удаленными объектами в реальном времени, соответственно Кс = 1,12. Тогда общая трудоемкость разработки определяется по формуле:

|  |  |
| --- | --- |
|  | (7.1) |

где – дополнительный коэффициент сложности;

– коэффициент, учитывающий использование типовых программ и модулей;

– коэффициент новизны.

Подставляя значения в формулу 7.1, получим:

(человеко-дней)

На основе общей трудоемкости определяется плановое число разработчиков по формуле:

|  |  |
| --- | --- |
|  | (7.2) |

где – численность исполнителей проекта;

– общая трудоемкость разработки проекта (чел./дн.);

– срок разработки проекта (лет);

– эффективный фонд времени работы одного работника в течение года (дн).

Срок разработки проекта составляет полгода ( = 0,5 год). Эффективный фонд времени определяется по формуле:

|  |  |
| --- | --- |
| , | (7.3) |

где – количество дней в году;

– количество праздничных дней в году;

– количество выходных дней в году;

– количество дней отпуска.

Подставляя значения в формулу 7.3, получим:

(дней).

При решении сложных задач с длительным периодом разработки ПО трудоемкость определяется по стадиям разработки: техническое задание (ТЗ), эскизный проект (ЭП), технический проект (ТП), рабочий проект (РП), внедрение (ВН). При этом трудоемкость разработки ПО отличается в зависимости от стадий. Общий вид формулы, по которой рассчитывается трудоемкость изготовления ПС:

|  |  |
| --- | --- |
| , | (7.4) |

где – трудоемкость изготовления ПО на данной стадии;

– нормативная трудоемкость;

– удельный вес трудоемкости выбранной стадии разработки ПО в общей трудоемкости разработки ПО.

При этом для стадии «Рабочий проект» полученное значение трудоемкости изготовления ПО требуется умножить на . Общая трудоемкость определяется как сумма трудоемкости изготовления ПО на каждой из стадий разработки. Результаты расчетов трудоемкости по стадиям сведены в таблицу 7.2.

Таблица 7.2 – Расчет общей трудоемкости разработки ПО и численности исполнителей с учетом стадий

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1. Коэффициенты удельных весов трудоемкости стадии разработки ПО () | 0,11 | 0,09 | 0,11 | 0,55 | 0,14 | 1,00 |
| 2. Коэффициент сложности ПО | 1,12 | 1,12 | 1,12 | 1,12 | 1,12 |  |
| 3. Коэффициент, учитывающий использование стандартных модулей |  |  |  | 0,80 |  |  |
| 4. Коэффициент, учитывающий новизну ПО () | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 |  |
| 5. Общая трудоемкость ПО (), чел./дн. | 77,37 | 63,30 | 77,37 | 309,48 | 98,47 | 625,99 |

На основе уточненной трудоемкости разработки ПО с использованием формулы 7.2 найдем общую численность разработчиков, которые требуются, чтобы вложиться в заданные сроки:

**7.2.2** Расчет сметы затрат и цены заказного ПО

Основная статья расходов на создание ПО является заработная плата разработчиков проекта, то есть людей, непосредственно занимающихся разработкой.

Месячная тарифная ставка каждого специалиста () определяется по формуле:

|  |  |
| --- | --- |
| , | (7.5) |

где – месячная тарифная ставка первого разряда (тыс.руб.);

– тарифный коэффициент, соответствующий установленному тарифному разряду.

Часовая тарифная ставка :

|  |  |
| --- | --- |
|  | (7.6) |

где – часовая тарифная ставка (тыс.руб);

– среднемесячная норма рабочего времени в часах (составляет 170 часов).

Расчет месячных и часовых тарифных ставок сведен в таблицу 7.3 с учётом того, что месячная тарифная ставка первого разряда на март 2016 года составляет 295 (тыс.руб.).

Таблица 7.3 – Расчет месячных и почасовых тарифных ставок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Должность | Количество ставок | Тарифный разряд | Тарифный коэффициент | Месячная тарифная ставка (руб.) | Часовая тарифная ставка (руб.) |
| Ведущий инженер-программист | 1 | 14 | 3,25 | 949000 | 5582,35 |
| Инженер-программист 1-ой категории | 5 | 12 | 2,84 | 829280 | 4878,12 |

Основная заработная плата исполнителей рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (7.7) |

где n – количество исполнителей;

– часовая тарифная ставка i-го исполнителя (ден.ед.);

– количество часов работы в день, ч;

– коэффициент премирования;

– плановый фонд рабочего времени i-го исполнителя (дн.).

(руб).

Дополнительная заработная плата () включает в себя оплаты отпусков и другие выплаты, предусмотренные законодательством, и определяется по формуле:

|  |  |
| --- | --- |
|  | (7.8) |

где – норматив дополнительной заработной платы (10-20%).

|  |  |
| --- | --- |
|  |  |

Отчисления в фонд социальной защиты определяется по формуле:

|  |  |
| --- | --- |
|  | (7.9) |

где – норматив отчислений в фонд социальной защиты наделения.

Отчисления в фонд социальной защиты – 34%, отчисления в фонд социального страхования – 0,6%. Исходя из этого, получаем:

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Материалы» отражают расходы на магнитные носители, бумагу, тонер и прочие вещи, необходимые для разработки ПО. Нормы расхода материалов в суммарном выражении () определяется в расчете на 100 строк исходного кода. Сумма затрат на расходные материалы определяется по формуле:

|  |  |
| --- | --- |
|  | (7.10) |

где – норма расхода материалов в расчете на 100 строк исходного кода ПО (руб);

– общий объем ПО (строк исходного кода).

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Машинное время» включает оплату машинного времени, необходимого для разработки и отладки ПО. Норматив на 100 строк исходного кода () зависит от характера решаемых задачи и типа приложений. Расходы по этой статье определяются по формуле:

|  |  |
| --- | --- |
| , | (7.11) |

где – цена одного машино-часа (руб);

– общее время работы над проектом (часов).

(руб).

Расходы по статье «Научные командировки» () определяются по формуле:

|  |  |
| --- | --- |
|  | (7.12) |

где – норматив расходов на командировки в целом по организации (10%).

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Прочие затраты» () включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяется по формуле:

|  |  |
| --- | --- |
|  | (7.13) |

где – норматив прочих затрат в целом по организации.

|  |  |
| --- | --- |
|  |  |

Затраты по статье «Накладные расходы» () связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |
| --- | --- |
|  | (7.14) |

где – норматив накладных расходов в целом по организации.

Общая сумма расходов по смете () определяется по формуле:

(руб).

Организация-разработчик участвует в освоении ПС и несет соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком. Для упрощения расчетов затраты на освоение ПС определяются по нормативу от себестоимости ПС в расчете на 3 месяца по формуле:

|  |  |
| --- | --- |
|  | (7.15) |

где Но – норматив расходов на освоение, Но = 10%.

Затраты на сопровождение и адаптацию ПО ():

|  |  |
| --- | --- |
|  | (7.16) |

где – норматив расходов на сопровождение (%).

Прибыль от создаваемого ПО определяется по формуле:

|  |  |
| --- | --- |
| , | (7.17) |

где – уровень рентабельности ПО (%).

Прогнозируемая цена ПО без налогов () определяется по формуле:

|  |  |
| --- | --- |
|  | (7.18) |

Подставляя значения в формулу 7.18, получим:

(руб).

Налог на добавленную стоимость ():

|  |  |
| --- | --- |
|  | (7.19) |

где – норматив НДС (%).

Прогнозируемая отпускная цена ():

|  |  |
| --- | --- |
| . | (7.20) |

Подставляя значения в формулу 7.20, получим:

(руб).

Все расчеты себестоимости и прибыли от реализации программного средства сведены в таблицу 7.4.

Таблица 7.4 – Результаты и формулы расчетов.

|  |  |  |
| --- | --- | --- |
| Наименование статей | Усл. обозн. | Значение (руб) |
| 1 | 2 | 3 |
| Основная заработная плата исполнителей | Зо | 36782793,41 |
| Дополнительная заработная плата исполнителей | Зд | 5517419,01 |
| Отчисления в фонд социальной защиты населения и фонд обязательного страхования | Зсз | 14635873,5 |
| Машинное время | Рм | 4248000 |
| Расходы на материалы | M | 7116,2 |
| Расходы на научные командировки | Рнк | 3678279,34 |
| Прочие прямые расходы | Пз | 7356558,68 |
| Накладные расходы | Рн | 18391396,71 |
| Полная себестоимость | Сп | 90617436,85 |
| Прогнозируемая прибыль | По | 27185231,06 |
| Прогнозируемая цена без налогов (цена предприятия) | Цп | 27185241,06 |
| Налог на добавленную стоимость (НДС) | НДС | 5437048,21 |
| Прогнозируемая отпускная цена | Цо | 32622289,27 |

где Нп – ставка налога на прибыль (Нп = 18%).

Подставляя значения в формулу 7.21, получим:

(руб). (7.21)

Данная сумма остается у собственника.

Все расчеты себестоимости и прибыли представлены в таблице 7.4.

Таким образом, в результате оценки экономического эффекта у разработчика были получены следующие результаты:

1. Cебестоимость проекта составила 90,62 миллиона рублей.
2. Прогнозируемая отпускная цена – 32,62 миллиона рублей.
3. Прибыль за вычетом налога составит 22,29 миллиона рублей.

**7.3** Расчет экономического эффекта от применения программного средства у пользователя

Для определения экономического эффекта от использования нового ПС у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПС (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям при использовании прежнего варианта ПС. При сравнении базового и нового вариантов ПС в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов относительно базового варианта. При этом создание нового ПС окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшие 2–3 года.

Исходные данные для определения экономического эффекта приведены в таблице 7.5.

Общие капитальные затраты для пользователя составят:

|  |  |
| --- | --- |
| , | (7.22) |

где Кпр – затраты пользователя на приобретение ПС по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации (руб.);

Кос  – затраты пользователя на освоение ПС (руб.);

Кс – затраты пользователя на оплату услуг по сопровождению ПС(руб.);

Ктс – изатраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПС (руб.);

Коб – затраты на пополнение оборотных средств в связи с использо­ванием нового ПС (руб.).

Ко = 32622289,27 + 9061743,69 + 22291889,47 + 2 500 000 =   
= 314 568 274 (руб.).

Экономия затрат на заработную плату (Сз) при использовании нового ПС в расчете на объем выполненных работ:

|  |  |
| --- | --- |
| , | (7.23) |

где Сзе – экономия затрат на заработную плату при решении задач c использованием нового ПС (руб.);

А2 – количество типовых задач, решаемых за год (задач).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 7.5 – Исходные данные для определения экономического эффекта | | | | |
| Наименование показателей | Обозна-чение | Единицы измер. | Значение показателя | |
| в базовом варианте | в новом варианте |
| Капитальные вложения, включая затраты пользователя на приобретение ПС | Кпр | руб. | - | 32622289,27 |
| Затраты на освоение ПС | Кос | руб. | - | 9061743,69 |
| Затраты на сопровождение ПС | Кс | руб. | - | 4530871,84 |
| Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПС | Ктс | руб. | - | 22291889,47 |

*Продолжение таблицы 7.5*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПС | Коб | руб. | - | 2500000 |
| Среднемесячная ЗП одного программиста | Зсм | руб. | 849233,33 |  |
| Коэффициент начислений на зарплату | Кнз |  | 1,3 |  |
| Среднемесячное количество рабочих дней | Др | день | 21 |  |
| Количество типовых задач, решаемых за год | Зт | задача | 20000 |  |
| Объем выполняемых работ за год | А1,А2 | задача | 20000 |  |
| Средняя трудоемкость работ | Тс1, Тс2 | чел.-час на задачу | 1,5 | 0,1 |
| Средний расход машинного времени | Мв1,Мв2 | маш.-час на задачу | 1,5 | 1,0 |
| Цена 1-го машино-часа работы ЭВМ | Цм | руб. | 4500 |  |
| Количество часов работы в день | Тч | ч | 8 |  |
| Ставка налога на прибыль | Нп | % | 18 |  |

Экономия затрат на заработную плату в расчете на 1 задачу рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.24) |

где Зсм – среднемесячная заработная плата одного программиста (руб.);

Тс1, Тс2 – трудоемкости работ (человеко-часов) в расчете на 1 задачу;

Тч  – количество часов работы в день (ч);

Др – среднемесячное количество рабочих дней.

Сзе = = 7076,94 (руб.).

Сз = 7076,94 ∙ 20 000 = 141538888,9 (руб.).

Экономия затрат за счет сокращения начислений на заработную плату (Сн) при коэффициенте начислений равном 1,5:

Сн = 141538888,9 ∙ 1,5 = 184000556 (руб.).

Экономия затрат на оплату машинного времени (См) в расчете на выполненный объем работ в результате применения нового ПС:

|  |  |
| --- | --- |
| , | (7.25) |

где Сме – экономия затрат на оплату машинного времени при решении задач с использованием нового ПС.

Экономия затрат на оплату машинного времени в расчете на 1 задачу:

|  |  |
| --- | --- |
|  | (7.26) |

где Цм – цена одного машино-часа работы ЭВМ;

Мв1, Мв2 – средний расход машинного времени при применении соответственно базового и нового ПС.

Сме = 4 500 ∙ (1.5 – 1) = 2250 (руб.)

См = 2250 ∙ 20 000 = 45000000 (руб.).

Общая годовая экономия текущих затрат, связанных с использованием нового ПС:

|  |  |
| --- | --- |
|  | (7.27) |

Со = 184000556 + 45000000 = 229000556 (руб.)

Внедрение нового ПС позволит пользователю сэкономить на текущих затратах 229000550руб., т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

|  |  |
| --- | --- |
|  | (7.28) |

где Нп – ставка налога на прибыль.

ΔПч = 229000556 – (29000556 ∙ 18) / 100 = 187780456 (руб.)

В процессе использования нового ПС чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капиталовложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2016 год) путем умножения результатов и затрат за каждый год на коэффициент приведения αt, который рассчитывается по формуле [16]:

|  |  |
| --- | --- |
|  | (7.29) |

где Ен – норматив приведения разновременных затрат и результатов, Ен=36%;

t – номер года, результаты и затраты которого приводятся к   
расчетному (2017 - 1, 2018 - 2, 2019 - 3 и т.д.);

tр  – расчетный год (2016).

Следовательно, при решении данной задачи коэффициентам приведения (αt) по годам будут соответствовать следующие значения:

αt0 = 1,000 – первый расчетный год;

αt1 = (1 + 0,36)2016-2017 = 0,735 – 2017 расчетный год;

αt2 = (1 + 0,36)2017-2018 = 0,541 – 2018 расчетный год;

αt3 = (1 + 0,36)2018-2019 = 0,398 – 2019 расчетный год.

Сведем данные расчета экономического эффекта в таблицу 7.6.

Таблица 7.6 – Расчет экономического эффекта от использования нового ПС

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
| Показатели | Ед. изм. | Годы | | | |
| 2016 | 2017 | 2018 | 2019 |
| Результаты: | | | | |  |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 187780456 | 187780456 | 187780456 |
| То же с учетом фактора времени | руб. |  | 138073864 | 101524900 | 74650662 |
| Затраты: | | | | | |
| Приобретение ПС (Кпр) | руб. | 32622289 |  |  |  |
| Освоение ПС (Кос) | руб. | 9061744 |  |  |  |
| Сопровождение (Кс) | руб. | 4530872 | 4530872 | 4530872 | 4530872 |
| Доукомплектование ВТ техническими средствами (Ктс) | руб. | 22291889 |  |  |  |
| Пополнение оборотных средств (Коб) | руб. | 2 500 000 | 2 500 000 | 2 500 000 | 2 500 000 |
| Всего затрат | руб. | 71006794 | 7030872 | 7030872 | 7030872 |
| То же с учетом фактора времени | руб. | 71006794 | 5169759 | 3801293 | 2795069 |
| Экономический эффект: | | | | | |
| Превышение результатов над затратами | руб. | -71006794 | 180749584 | 180749584 | 180749584 |
| То же нарастающим итогом | руб. | -71006794 | 132904106 | 97723607 | 7185594 |
| Коэффициент приведения | ед. | 1 | 0.735 | 0.541 | 0398 |

Таким образом, реализация проекта ПС оптимизации обслуживания клиентов на высоконагруженном сервере позволит снизить трудоемкость решения задач обслуживания клиентов и сократить количество использующихся ЭВМ. Все затраты заказчика окупятся в третьем году эксплуатации ПС. Проект представляется эффективным и полезным для заказчика.

Индекс рентабельности проекта (Ри) показывает эффективность проекта по уровню доходов на единицу затрат. Рассчитывается по формуле [17]:

|  |  |
| --- | --- |
|  | (7.30) |

где Пчt – прибыль за определенный промежуток времени (руб.);

Коt – затраты на ПС за определенный промежуток времени (руб.).

В результате расчетов получаем:

* чистый дисконтированный доход в 2016 году равен 132904106 рублей;
* программное средство окупится на втором году;
* коэффициент рентабельности составил 3,80.

Таким образом, разработка и применение программного продукта является экономически целесообразной.

ЗАКЛЮЧЕНИЕ

В процессе работы над дипломным проектом были рассмотрены основные принципы построения модулей анализа изображений. В результате было создан полноценный модуль анализа качества зерна. По результатам этого, было создано приложение, не имеющее полноценных аналогов. Используя данный модуль, пользователи получат следующие преимущества:

* увеличение дружественности пользовательского интерфейса;
* повышение скорости и эффективности анализа качества зерна

Программный модуль имеет следующие достоинства:

* простая интеграция новых методов анализа и фильтрации изображения;
* высокая скорость анализа;
* простота работы для неквалифицированного рабочего;

В качестве недостатка программного модуля расширения можно отметить достаточно длительный подсчет признаков и зависимость качества анализа от обучающего изображения.

В качестве способов улучшения можно рассматривать добавление новых типов фильтрации изображения, что способно повысить автономность модуля.

Список использованных источников

1. Иванова, Т.Н. Товароведение и экспертиза зерномучных товаров / Т.Н. Иванова. - М.: Академия, 2004. – 288 с.
2. Зерно. Методы определения запаха и цвета (ГОСТ 10967 – 90) [Электронный ресурс]. – 2016. – Режим доступа : <http://docs.cntd.ru/document/gost-10967-90>.
3. Зерно. Методы определения зараженности и поврежденности вредителями (ГОСТ 13586.4 – 83) [Электронный ресурс]. – 2016. – Режим доступа : <http://docs.cntd.ru/document/1200024347>.
4. GRANOLYSER [Электронный ресурс] / GRANOLYSER – 2016. Режим доступа: http://www.pfeuffer.com/granolyser.html?&L=4.
5. Николаева М.А. Товарная экспертиза / М.А. Николаева. - М.: Экономика, 1998. – 288 с.
6. Зерно. Методы определения типового состава (ГОСТ 10940 – 64) [Электронный ресурс]. – 2016. – Режим доступа : http://docs.cntd.ru/document/gost-10940-64.
7. Волкова Л.Д. Товароведение и экспертиза продовольственных товаров. Раздел Зерномучные товары / Л.Д. Волкова, В.И. Заикина, С.С. Гурьянова. - М.: МУПК, 1999. – 63 с.
8. Цветовая модель [Электронный ресурс] / Цветовая модель – 2016. Режим доступа: http://mikhalkevich.narod.ru/kyrs/Cvetovedenie/main1.html.
9. Морфологические операции [Электронный ресурс] / Морфологические операции – 2016. Режим доступа: http://www.intuit.ru/studies/courses/10621/1105/lecture/17989?page=4.
10. Цифровые фильтры [Электронный ресурс] / Цифровые фильтры – 2016. Режим доступа: <http://sernam.ru/tau_42.php>.
11. Классификаторы [Электронный ресурс] / Классификаторы – 2016. Режим доступа: https://www.lektorium.tv/course/22780
12. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М. : Вильямс, 2012. – 1248 с.
13. MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа : https://msdn.microsoft.com/.
14. Qt creator [Электронный ресурс] / Qt creator – 2016. Режим доступа: https://www.qt.io/ru/download-open-source/.
15. OpenCV [Электронный ресурс] / OpenCV – 2016. Режим доступа: <http://opencv.org//>.
16. Тарифы в Беларуси [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.tarify.by.
17. Палицын В. А. Технико-экономическое обоснование дипломных проектов. Методическое пособие в 4-х частях. Часть 4. Проекты программных средств – Минск: БГУИР, 2005. – 21 с.

ПРИЛОЖЕНИЕ А

(*обязательное*)

Исходный текст типа SVMclassifier

SVMclassifier::SVMclassifier(QVector<Seed> seedVector, QVector<int> featVector, int clusters, QVector<QVector<int> > trainDataObj) : QObject(0)

{

seedVect = QVector<Seed>(seedVector);

featVect = QVector<int>(featVector);

trainigDataObjs = QVector<QVector<int> >(trainDataObj);

}

void SVMclassifier::FillTrainingMat()

{

// AREA = 0, PERIMETR, COMPACTNESS, ELONGATION, LUMA, CONTRAST, HOMOGENEITY, DISSIMILARITY, ENERGY, ENTROPY, CORRELATION, MATEXPECT, DISPERTION

int indexOfObject = 0;

int countOfOb = 0;

for(int i = 0; i < trainigDataObjs.length(); i++)

for(int j = 0; j < trainigDataObjs[i].length(); j++)

countOfOb++;

float labels[countOfOb];

float trainingData[countOfOb][featVect.length()]; //= new float\*[countOfOb];//[featVect.length()]

int k = 0;

for(int i = 0; i < trainigDataObjs.length(); i++)

for(int j = 0; j < trainigDataObjs[i].length(); j++)

{

labels[indexOfObject++] = i;

fillObject(trainingData[k++], trainigDataObjs[i][j]);

}

Mat labelsMat(countOfOb, 1, CV\_32FC1, labels);

Mat trainingDataMat(countOfOb, featVect.length(), CV\_32FC1, trainingData);

// Set up SVM's parameters

CvSVMParams params;

params.svm\_type = CvSVM::C\_SVC;

params.kernel\_type = CvSVM::LINEAR;

params.term\_crit = cvTermCriteria(CV\_TERMCRIT\_ITER, 100, 1e-6);

// Train the SVM

CvSVM SVM;

SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);

int featureLength = featVect.length();

float object[featureLength];

float confidence;

for(int i = 0; i < seedVect.length(); i++)

{

fillObject(object, i);

Mat objectMat(1, featureLength, CV\_32FC1, object);

float cl2 = SVM.predict(objectMat, true);

confidence = 1.0 / (1.0 + exp(-cl2));

float cl = SVM.predict(objectMat);

seedVect[i].SetCluster(cl);

seedVect[i].probability = confidence;

}

// for(int i =0; i<seedVect.length(); i++)

// cout << "area = " <<seedVect[i].GetArea() <<" luma = " <<seedVect[i].GetLuma() << " matexp = " << seedVect[i].matExpect

// <<" centre = " <<seedVect[i].centerMass << " ellong = " << seedVect[i].elongation << " cluster = " <<seedVect[i].GetCluster() <<"\n ";

for(int i =0; i<seedVect.length(); i++)

cout << seedVect[i].GetCluster() << ", ";

}

void SVMclassifier::fillObject(float \*arr, int numberOfSeed)

{

int l = 0;

if((featVect.indexOf(AREA)) != -1)

arr[l++] = seedVect[numberOfSeed].GetArea();

if((featVect.indexOf(COMPACTNESS)) != -1)

arr[l++] = seedVect[numberOfSeed].GetCompactness();

if((featVect.indexOf(PERIMETR)) != -1)

arr[l++] = seedVect[numberOfSeed].GetPerimetr();

if((featVect.indexOf(ELONGATION)) != -1)

arr[l++] = seedVect[numberOfSeed].elongation;

if((featVect.indexOf(LUMA)) != -1)

arr[l++] = seedVect[numberOfSeed].GetLuma();

if((featVect.indexOf(CONTRAST)) != -1)

arr[l++] = seedVect[numberOfSeed].contrast;

if((featVect.indexOf(HOMOGENEITY)) != -1)

arr[l++] = seedVect[numberOfSeed].homogeneity;

if((featVect.indexOf(DISSIMILARITY)) != -1)

arr[l++] = seedVect[numberOfSeed].dissimilarity;

if((featVect.indexOf(ENERGY)) != -1)

arr[l++] = seedVect[numberOfSeed].energy;

if((featVect.indexOf(ENTROPY)) != -1)

arr[l++] = seedVect[numberOfSeed].entropy;

if((featVect.indexOf(CORRELATION)) != -1)

arr[l++] = seedVect[numberOfSeed].correlation;

if((featVect.indexOf(MATEXPECT)) != -1)

arr[l++] = seedVect[numberOfSeed].matExpect;

if((featVect.indexOf(DISPERTION)) != -1)

arr[l++] = seedVect[numberOfSeed].dispersion;

}

QVector<Seed> SVMclassifier::GetSeedVector()

{

return seedVect;

}

ПРИЛОЖЕНИЕ Б

(*обязательное*)

Исходный текст типа FeaturesCalculation

FeaturesCalculation::FeaturesCalculation(Mat img, QVector<Seed> seedVector, Mat first)

{

srcImg = img.clone();

firstImg = first.clone();

seedVect = QVector<Seed>(seedVector);

calculateTextureGLCM();

}

FeaturesCalculation::~FeaturesCalculation()

{

}

enum PARAMETR { AREA, LUMA, MATEXP, MASSCENTRE, ELLONGATION, CONTRAST, HOMOGENEITY, DISSIMILARITY, ENERGY, ENTROPY, CORRELATION };

void FeaturesCalculation::calculateSomeGeometryParam(PARAMETR param)

{

int i = 0, oldArea, oldLuma, oldPixel;

int len = seedVect.length();

Mat channel[3], YCRImage;

cvtColor(firstImg, YCRImage, COLOR\_BGR2YCrCb);

split(YCRImage, channel);

double \*m11 = new double[len];

double \*m02 = new double[len];

double \*m20 = new double[len];

for(int i =0; i < len; i++)

{

m11[i] = 0;

m02[i] = 0;

m20[i] = 0;

}

for( int y = 0; y < srcImg.rows; y++ )

{

for( int x = 0; x < srcImg.cols; x++ )

{

if(!(srcImg.at<Vec3b>(y,x)[0] == 0 &&

srcImg.at<Vec3b>(y,x)[1] == 0 &&

srcImg.at<Vec3b>(y,x)[2] == 0))

{

i = 0;

for(Seed s: seedVect)

{

if(s.GetColor().val[0] == srcImg.at<Vec3b>(y,x)[0] &&

s.GetColor().val[1] == srcImg.at<Vec3b>(y,x)[1] &&

s.GetColor().val[2] == srcImg.at<Vec3b>(y,x)[2])

{

if(param == AREA)

{

oldArea = s.GetArea();

seedVect[i].SetArea(oldArea+1);

break;

}

if(param == LUMA)

{

oldLuma = s.GetLuma();

int l = channel[0].at<uchar>(y, x);

seedVect[i].SetLuma(oldLuma + l);

oldPixel = s.GetCountOfPixels();

seedVect[i].SetCountOfPixels(oldPixel+1);

break;

}

if(param == MATEXP)

{

seedVect[i].countOfPixelsOnLevel[channel[0].at<uchar>(y,x)]++;

}

if(param == MASSCENTRE)

{

seedVect[i].centerMass.x += x ;

seedVect[i].centerMass.y += y ;

}

if(param == ELONGATION)

{

m11[i] += (x - seedVect[i].centerMass.x)\*(y - seedVect[i].centerMass.y);

m02[i] += qPow((y - seedVect[i].centerMass.y), 2);

m20[i] += qPow((x - seedVect[i].centerMass.x), 2);

}

}

i++;

}

}

}

}

if(param == LUMA)

{

float luma = 0;

for(int k = 0; k < seedVect.length(); k++)

{

luma = seedVect[k].GetLuma() / seedVect[k].GetCountOfPixels();

seedVect[k].SetLuma(luma);

luma = 0;

}

}

if(param == MATEXP)

{

for(int index = 0; index < seedVect.length(); index++)

{

for(int k = 0; k < 256; k++)

{

seedVect[index].matExpect += k \* seedVect[index].countOfPixelsOnLevel[k] / seedVect[index].GetArea();

}

}

}

if(param == MASSCENTRE)

{

for(int index = 0; index < seedVect.length(); index++)

{

seedVect[index].centerMass.x /= seedVect[index].GetArea();

seedVect[index].centerMass.y /= seedVect[index].GetArea();

}

}

if(param == ELONGATION)

{

double m1=0, m2=0;

for(int index = 0; index < seedVect.length(); index++)

{

m1 = (m20[index] + m02[index] + qSqrt((m20[index] - m02[index])\*(m20[index] - m02[index]) + 4\*m11[index]\*m11[index]));

m2 = (m20[index] + m02[index] - qSqrt((m20[index] - m02[index])\*(m20[index] - m02[index]) + 4\*m11[index]\*m11[index]));

}

}

}

void FeaturesCalculation::calculateArea()

{

calculateSomeGeometryParam(AREA);

}

void FeaturesCalculation::calculatePerimetr()

{

int oldPerimetr, i;

for( int y = 0; y < srcImg.rows; y++ )

{

for( int x = 0; x < srcImg.cols; x++ )

{

if(!(srcImg.at<Vec3b>(y,x)[0] == 0 &&

srcImg.at<Vec3b>(y,x)[1] == 0 &&

srcImg.at<Vec3b>(y,x)[2] == 0))

{

if(HaveBlackNeighbors( x, y))

{

i = 0;

for(Seed s: seedVect)

{

if(s.GetColor().val[0] == srcImg.at<Vec3b>(y,x)[0] &&

s.GetColor().val[1] == srcImg.at<Vec3b>(y,x)[1] &&

s.GetColor().val[2] == srcImg.at<Vec3b>(y,x)[2])

{

oldPerimetr = s.GetPerimetr();

seedVect[i].SetPerimetr(oldPerimetr+1);

break;

}

i++;

}

}

}

}

}

}

void FeaturesCalculation::calculateCompactness()

{

double compact = 0;

for(int i = 0; i < seedVect.length(); i++)

{

compact = seedVect[i].GetPerimetr()\*seedVect[i].GetPerimetr() / seedVect[i].GetArea();

seedVect[i].SetCompactness(compact);

}

}

bool FeaturesCalculation::HaveBlackNeighbors( int x, int y)

{

if((srcImg.at<Vec3b>(y-1,x)[0] == 0 &&

srcImg.at<Vec3b>(y-1,x)[1] == 0 &&

srcImg.at<Vec3b>(y-1,x)[2] == 0) ||

(srcImg.at<Vec3b>(y+1,x)[0] == 0 &&

srcImg.at<Vec3b>(y+1,x)[1] == 0 &&

srcImg.at<Vec3b>(y+1,x)[2] == 0) ||

(srcImg.at<Vec3b>(y,x-1)[0] == 0 &&

srcImg.at<Vec3b>(y,x-1)[1] == 0 &&

srcImg.at<Vec3b>(y,x-1)[2] == 0) ||

(srcImg.at<Vec3b>(y,x+1)[0] == 0 &&

srcImg.at<Vec3b>(y,x+1)[1] == 0 &&

srcImg.at<Vec3b>(y,x+1)[2] == 0))

{

return true;

}

else

{

return false;

}

}

QVector<Seed> FeaturesCalculation::GetSeedVector()

{

return seedVect;

}

void FeaturesCalculation::calculateLumaParameter(Mat srcImage)

{

calculateSomeGeometryParam(LUMA);

}

void FeaturesCalculation::calculateMatExpectation()

{

calculateSomeGeometryParam(MATEXP);

}

void FeaturesCalculation::calculateDispertion()

{

for(int index = 0; index < seedVect.length(); index++)

{

for(int k = 0; k < 256; k++)

{

seedVect[index].dispersion += qPow((k - seedVect[index].matExpect),2) \* seedVect[index].countOfPixelsOnLevel[k] / seedVect[index].GetArea();

}

}

}

void FeaturesCalculation::calculateMassCenter()

{

calculateSomeGeometryParam(MASSCENTRE);

}

void FeaturesCalculation::calculateElongation()

{

calculateSomeGeometryParam(ELONGATION);

}

void FeaturesCalculation::calculateTextureGLCM()

{

int leftP = 0, rightP = 0;

int count;

Mat gray = Mat::zeros(srcImg.size(), CV\_8UC1);

cvtColor(firstImg, gray, CV\_BGR2GRAY);

for( int y = 0; y < srcImg.rows; y++ )

{

for( int x = 0; x < srcImg.cols; x++ )

{

//printf("p(%d,%d)[%d][%d][%d]\n",y,x, srcImg.at<Vec3b>(y,x)[0],srcImg.at<Vec3b>(y,x)[1],srcImg.at<Vec3b>(y,x)[2]);

if(!(srcImg.at<Vec3b>(y,x)[0] == 0 &&

srcImg.at<Vec3b>(y,x)[1] == 0 &&

srcImg.at<Vec3b>(y,x)[2] == 0))

{

int i = 0;

for(Seed s: seedVect)

{

if((s.GetColor().val[0] == srcImg.at<Vec3b>(y,x)[0] &&

s.GetColor().val[1] == srcImg.at<Vec3b>(y,x)[1] &&

s.GetColor().val[2] == srcImg.at<Vec3b>(y,x)[2]) &&

!(srcImg.at<Vec3b>(y,x+1)[0] == 0 &&

srcImg.at<Vec3b>(y,x+1)[1] == 0 &&

srcImg.at<Vec3b>(y,x+1)[2] == 0 ))

{

leftP = gray.at<uchar>(y,x)/8;

rightP = gray.at<uchar>(y,x+1)/8;

count = seedVect[i].GLCM.at<uchar>(leftP, rightP);

count++;

seedVect[i].GLCM.at<uchar>(leftP, rightP) = count;

}

i++;

}

}

}

}

for(int k = 0; k < seedVect.length(); k++)

{

for(int i=0; i< seedVect[k].GLCM.rows; i++)

for(int j=0; j< seedVect[k].GLCM.cols; j++)

{

if(seedVect[k].GLCM.at<uchar>(i,j) != 0)

seedVect[k].countOfPairs++;

}

}

for(int k = 0; k < seedVect.length(); k++)

{

createGLCM(k);

}

}

void FeaturesCalculation::createGLCM(int indexOfSeed)

{

if(!srcImg.empty())

{

Mat GLCMtrasposed = Mat::zeros(32, 32, CV\_8UC1);

Mat GLCMsymmetric = Mat::zeros(32, 32, CV\_8UC1);

imshow("glcm1", seedVect[indexOfSeed].GLCM);

for(int i=0; i< seedVect[indexOfSeed].GLCM.rows; i++)

for(int j=0; j< seedVect[indexOfSeed].GLCM.cols; j++)

{

GLCMtrasposed.at<uchar>(j,i) = seedVect[indexOfSeed].GLCM.at<uchar>(i,j);

}

for(int i=0; i<seedVect[indexOfSeed].GLCM.rows; i++)

for(int j=0; j<seedVect[indexOfSeed].GLCM.cols; j++)

{

GLCMsymmetric.at<uchar>(i,j) = (seedVect[indexOfSeed].GLCM.at<uchar>(i,j) + GLCMtrasposed.at<uchar>(i,j));// countOfPairs;

}

seedVect[indexOfSeed].GLCM = GLCMsymmetric;

}

}

void FeaturesCalculation::calculateContrast()

{

calculateTextureParameter(CONTRAST);

}

void FeaturesCalculation::calculateHomogeneity()

{

calculateTextureParameter(HOMOGENEITY);

}

void FeaturesCalculation::calculateDissimilarity()

{

calculateTextureParameter(DISSIMILARITY);

}

void FeaturesCalculation::calculateEntropy()

{

calculateTextureParameter(ENTROPY);

}

void FeaturesCalculation::calculateEnergy()

{

calculateTextureParameter(ENERGY);

}

void FeaturesCalculation::calculateCorrelation()

{

calculateTextureParameter(CORRELATION);

}

void FeaturesCalculation::calculateTextureParameter(PARAMETR param)

{

float qrt = 0, fabs = 0, thigmaSqr = 0, U = 0;

for(int k = 0; k < seedVect.length(); k++)

{

for(int i=0; i< seedVect[k].GLCM.rows; i++)

for(int j=0; j< seedVect[k].GLCM.cols; j++)

{

if(param == DISSIMILARITY)

{

qrt = qPow(i-j,2);

fabs = qSqrt(qrt);

seedVect[k].dissimilarity += fabs\*seedVect[k].GLCM.at<uchar>(i,j);

}

if(param == ENERGY)

{

seedVect[k].energy += qPow(seedVect[k].GLCM.at<uchar>(i,j),2);

}

if(param == ENTROPY)

{

if (seedVect[k].GLCM.at<uchar>(i,j) != 0)

{

float ln = -qLn(seedVect[k].GLCM.at<uchar>(i,j));

seedVect[k].entropy += -qLn(seedVect[k].GLCM.at<uchar>(i,j)) \* seedVect[k].GLCM.at<uchar>(i,j);

}

printf("ln = %f, char =%d entropy = %f\n", ln, seedVect[k].GLCM.at<uchar>(i,j), seedVect[k].entropy);

}

if(param == HOMOGENEITY)

{

qrt = qPow(i-j,2);

if(qrt != 1)

seedVect[k].homogeneity += (1/(1-qPow(i-j,2)))\*seedVect[k].GLCM.at<uchar>(i,j);

}

if(param == CONTRAST)

{

int pixel = seedVect[k].GLCM.at<uchar>(i,j);

float dividing = pixel;

float qrt = qPow(i-j, 2);

float contr = qrt \* dividing;

seedVect[k].contrast += contr;

}

if(param == CORRELATION)

{

calculateUandThigma(k, U, thigmaSqr);

seedVect[k].correlation += seedVect[k].GLCM.at<uchar>(i,j)\*(i-U)\*(j-U) / thigmaSqr;

}

}

if(param == DISSIMILARITY)

seedVect[k].dissimilarity = seedVect[k].dissimilarity / (seedVect[k].countOfPairs);

if(param == HOMOGENEITY)

seedVect[k].homogeneity = seedVect[k].homogeneity / (seedVect[k].countOfPairs);

if(param == CONTRAST)

seedVect[k].contrast = seedVect[k].contrast/(seedVect[k].countOfPairs);

if(param == ENERGY)

seedVect[k].energy = seedVect[k].energy / (seedVect[k].countOfPairs);

if(param == ENTROPY)

seedVect[k].entropy = seedVect[k].entropy / (seedVect[k].countOfPairs);

if(param == CORRELATION)

seedVect[k].correlation = seedVect[k].correlation / (seedVect[k].countOfPairs);

}

}

void FeaturesCalculation::calculateUandThigma(int index, float &U, float &thigmaSqr)

{

for(int i=0; i< seedVect[index].GLCM.rows; i++)

for(int j=0; j< seedVect[index].GLCM.cols; j++)

{

U += seedVect[index].GLCM.at<uchar>(i,j)\*i;

}

U = U / seedVect[index].countOfPairs;

for(int i=0; i< seedVect[index].GLCM.rows; i++)

for(int j=0; j< seedVect[index].GLCM.cols; j++)

{

thigmaSqr += seedVect[index].GLCM.at<uchar>(i,j)\*(i-U)\*(i-U);

}

thigmaSqr = thigmaSqr / seedVect[index].countOfPairs;

}

ПРИЛОЖЕНИЕ В

(*обязательное*)

Исходный текст типа ROCDialog

ROCDialog::ROCDialog(QString fileName, QVector<Seed> seedVect, QWidget \*parent) :

QDialog(parent),

ui(new Ui::ROCDialog)

{

ui->setupUi(this);

this->fileName = QString(fileName);

seedVector = QVector<Seed>(seedVect);

this->TP = QVector<double>();

this->FP = QVector<double>();

}

ROCDialog::~ROCDialog()

{

delete ui;

}

void ROCDialog::drawRocCurve(int posclass)

{

double labels[27] = { 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 };

float MAXprobability = 0, MINprobability = seedVector[0].probability;

int n = seedVector.length();

float N=0, P=0;

for(int i = 0; i < n ; i++) {

if(labels[i] == posclass ) P++;

else N++;

}

if( N == 0 || P == 0 )

cout << "I only found class 1 in the labels vector ...\n";

for( int i = 0; i < n; i++ )

{

if(MAXprobability < seedVector[i].probability)

MAXprobability = seedVector[i].probability;

if(MINprobability > seedVector[i].probability)

MINprobability = seedVector[i].probability;

}

float t = MINprobability;

float TP = 0, FP = 0;

do

{

TP = 0, FP = 0;

for(int i = 0; i < n; i++)

{

if(seedVector[i].probability >= t)

{

if(seedVector[i].GetCluster() == posclass)

TP++;

else

FP++;

}

}

float tp = TP/P, fp = FP/N;

this->TP.push\_back(tp);

this->FP.push\_back(fp);

t += 0.01;

} while(t < MAXprobability);

std::reverse(this->TP.begin(), this->TP.end());

std::reverse(this->FP.begin(), this->FP.end());

for(int j = 0; j < this->TP.length(); j++)

{

cout << this->FP[j] << " " << this->TP[j] << "\n";

}

makePlot();

}

void ROCDialog::calculateROCparemeters()

{

int \*idealClusterData = new int[seedVector.length()];

if(fileName.count("sds.png") > 0)

{

int idealClusterDataStatic[27] = { 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 };

for(int i = 0; i < seedVector.length(); i++)

{

idealClusterData[i] = idealClusterDataStatic[i];

}

}

if(fileName.count("lampSeed.png") > 0)

{

int idealClusterDataStatic[81] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1};

for(int i = 0; i < seedVector.length(); i++)

{

idealClusterData[i] = idealClusterDataStatic[i];

}

}

if(fileName.count("sds.png") > 0 || fileName.count("lampSeed.png") > 0)

{

ROCforCluster(idealClusterData);

this->show();

}

}

void ROCDialog::ROCforCluster(int \*labels)

{

int n = seedVector.length();

double scores[n];

double scores2[n];

for(int j = 0; j < n; j++)

{

scores[j] = seedVector[j].probability;

scores2[j] = 1 - seedVector[j].probability;

}

calcAUC(labels, scores, seedVector.length(), 0);

calcAUC(labels, scores2, seedVector.length(), 1);

makePlot();

}

double ROCDialog::calcAUC(int\* labels, double \* scores,int n,int posclass)

{

typedef QPair<float,int> mypair;

QVector<mypair> L(n);

for(int i = 0; i < n; i++) {

L[i].first = scores[i];

L[i].second = labels[i];

}

qSort(L.begin(),L.end());

std::reverse(L.begin(), L.end());

/\* Count number of positive and negative examples first \*/

float N=0, P=0;

for(int i = 0; i < n ; i++) {

if(labels[i] == posclass ) P++;

else N++;

}

if( N == 0 || P == 0 )

cout << "I only found class 1 in the labels vector ...\n";

/\* Then calculate the actual are under the ROC curve \*/

double A = 0;

double fprev = INT\_MIN; //-infinity

double FP = 0,

TP = 0,

FPprev = 0,

TPprev = 0;

for(int i = 0 ; i < n; i++) {

double fi = L[i].first;

double label= L[i].second;

if(fi != fprev) {

/\* Divide area here already : a bit slower, but gains in precision and avoids overflows \*/

A = A + (trapezoidArea(FP\*1.0/N,FPprev\*1.0/N,TP\*1.0/P,TPprev\*1.0/P));

fprev = fi;

FPprev = FP;

TPprev = TP;

if(posclass == 0)

{

this->TP.push\_back(TP/P);

this->FP.push\_back(FP/N);

}

else {

this->TP2.push\_back(TP/P);

this->FP2.push\_back(FP/N);

}

}

if(label == posclass)

TP = TP + 1;

else

FP = FP + 1;

}

if(posclass == 0)

{

this->TP.push\_back(TP/P);

this->FP.push\_back(FP/N);

}

else {

this->TP2.push\_back(TP/P);

this->FP2.push\_back(FP/N);

}

QVector<QPair<float, float> > a = QVector<QPair<float, float> >(this->TP.length());

for(int j = 0; j < this->TP.length(); j++)

{

if(posclass == 0)

{

a[j].first = this->TP[j];

a[j].second = this->FP[j];

}

else {

a[j].first = this->TP2[j];

a[j].second = this->FP2[j];

}

}

smoothing(a);

qSort(a.begin(),a.end());

std::reverse(a.begin(), a.end());

if(posclass == 0)

{

this->a1 = QVector<QPair<float, float> > (a);

} else {

this->a2 = QVector<QPair<float, float> > (a);

}

if(posclass == 0)

{

this->TP.clear();

this->FP.clear();

}

else {

this->TP2.clear();

this->FP2.clear();

}

for(int j = 0; j < a.length(); j++)

{

if(posclass == 0)

{

this->TP.push\_back(a[j].first);

this->FP.push\_back(a[j].second);

}

else {

this->TP2.push\_back(a[j].first);

this->FP2.push\_back(a[j].second);

}

}

A = A + trapezoidArea(1.0,FPprev\*1.0/N,1.0,TPprev\*1.0/P);

return A;

}

double ROCDialog::trapezoidArea(double X1, double X2, double Y1, double Y2) {

double base = std::abs(X1-X2);

double height = (Y1+Y2)/2.0;

return (base \* height);

}

void ROCDialog::smoothing(QVector<QPair<float, float> > &a)

{

//How many neighbours to smooth

int NO\_OF\_NEIGHBOURS=3;

QVector<QPair<float, float> > tmp=a;

for(int i=0;i<a.size()-1;i++)

{

if(i+NO\_OF\_NEIGHBOURS+1<a.size())

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += a[i+j].first;

a[i].second += a[i+j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

}

else

{

for(int j=1;j<NO\_OF\_NEIGHBOURS;j++)

{

a[i].first += tmp[i-j].first;

a[i].second += tmp[i-j].second;

}

a[i].first /= NO\_OF\_NEIGHBOURS;

a[i].second /= NO\_OF\_NEIGHBOURS;

}

}

for(int i=1;i<a.size()-1;i++)

{

if(a[i].first < a[i-1].first)

a[i].first = a[i-1].first+0.1;

if(a[i].second < a[i-1].second)

a[i].second = a[i-1].second+0.1;

}

}

Приложение Г

(*обязательное*)

Спецификация программного дипломного проекта

Приложение Д

(*обязательное*)

Ведомость документов