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 последовательностей 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;

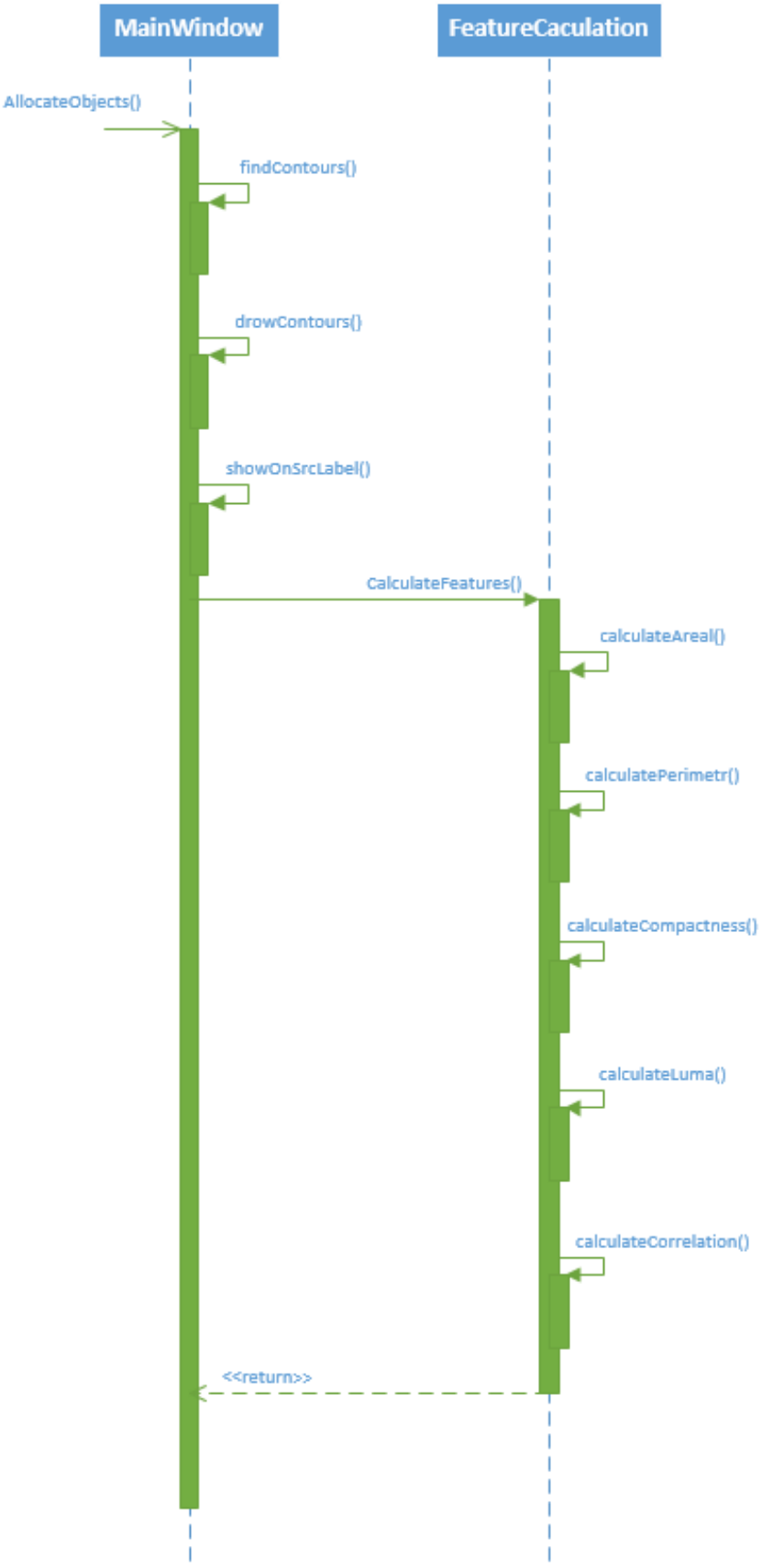

Рисунок 3.2 – Диаграмма последовательностей метода `AllocateObjects()`

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

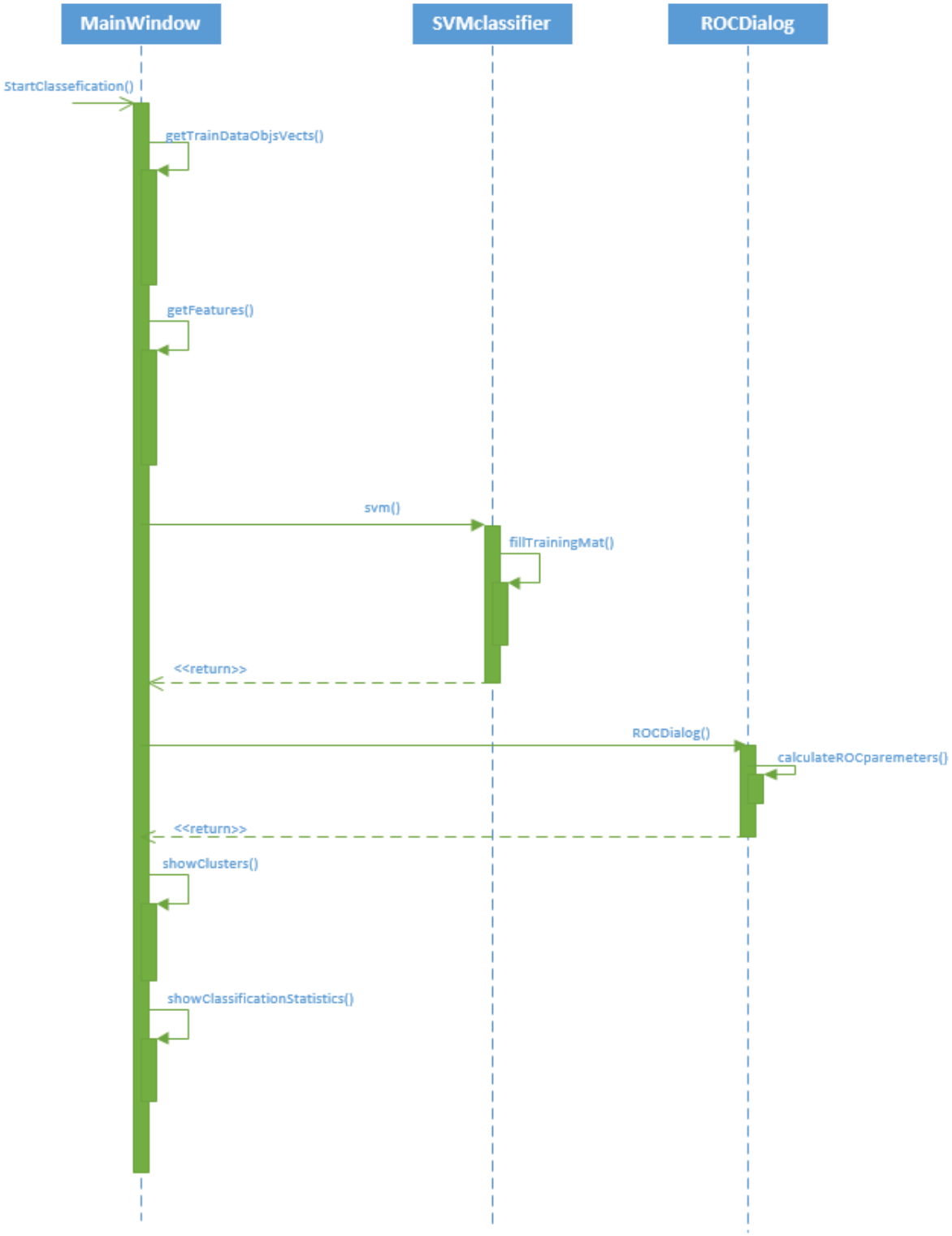

Рисунок 3.3 – Диаграмма последовательностей метода `StartClassification()`

– 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;

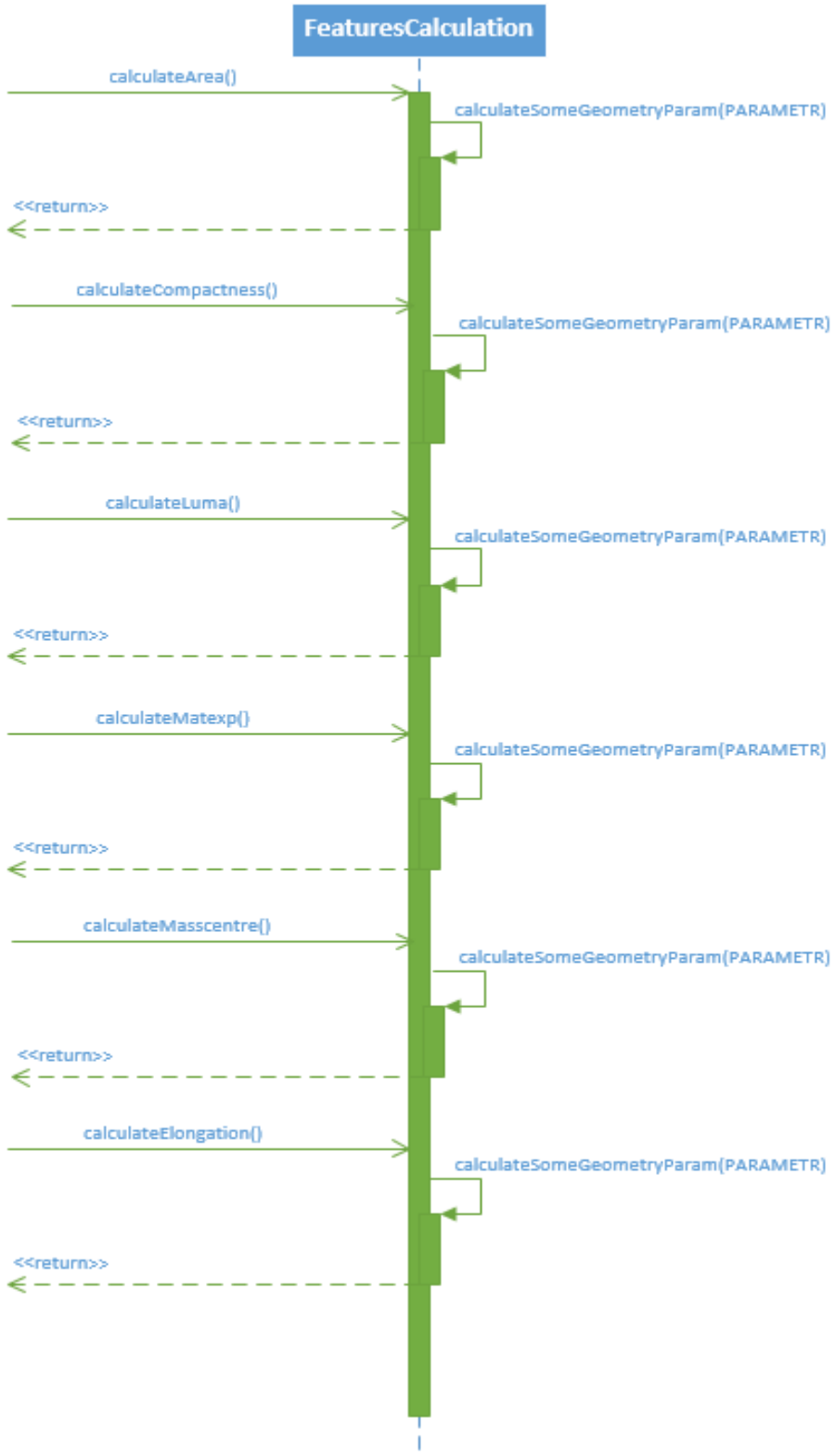

Рисунок 3.4 – Диаграмма последовательностей метода `calculateSomeGeometryParam()`

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

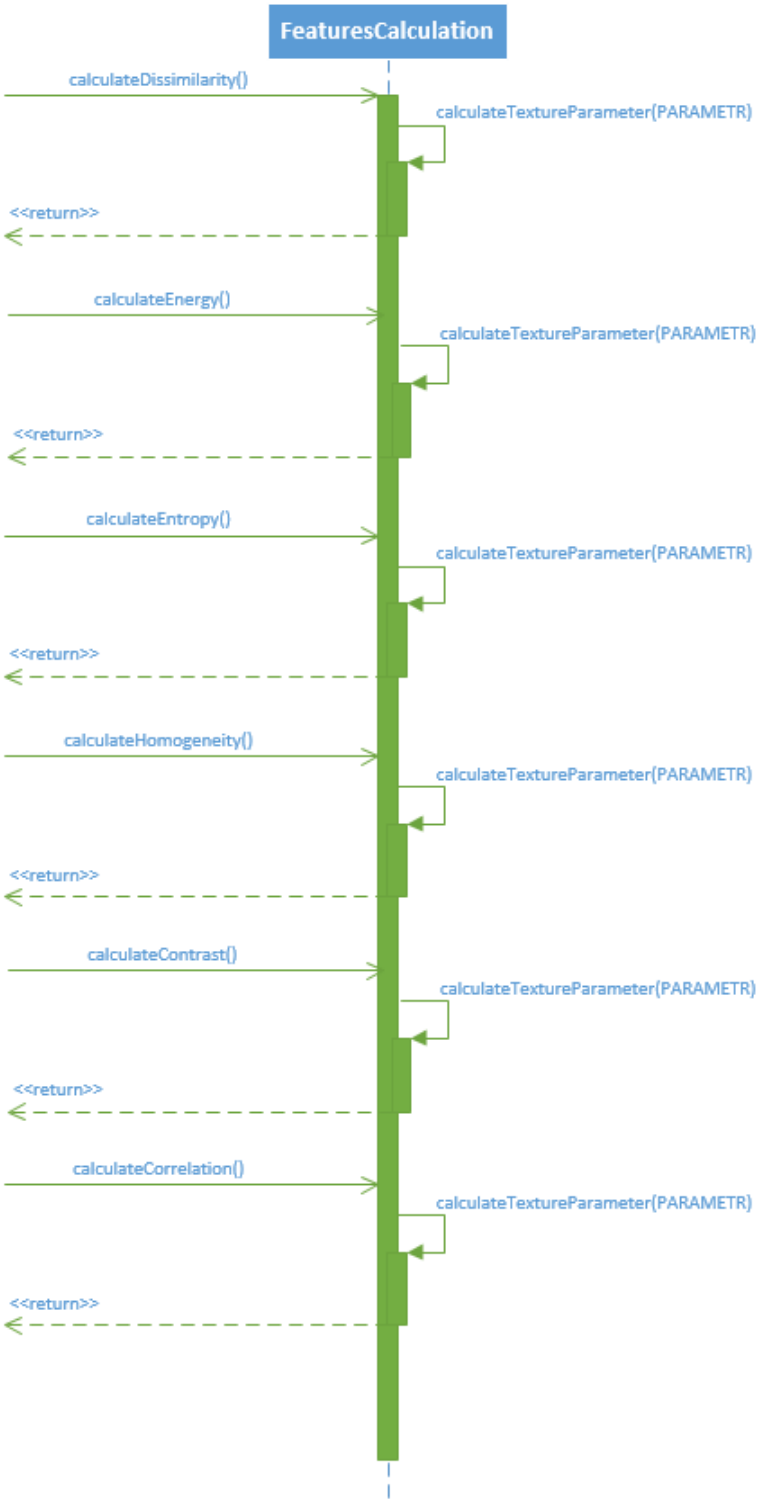

Рисунок 3.5 – Диаграмма последовательностей метода `calculateTextureParameter()`

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`.

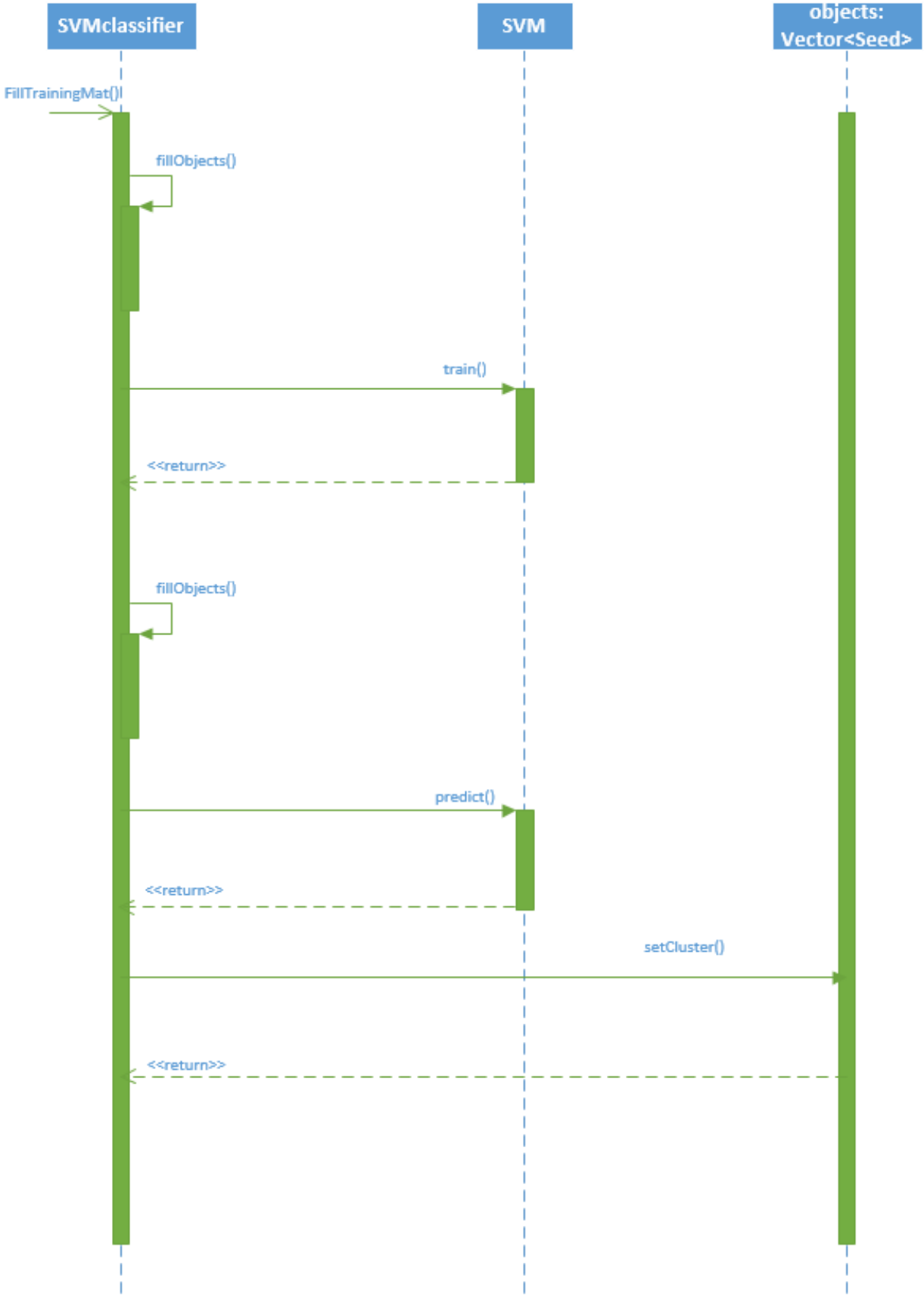

Рисунок 3.6 – Диаграмма последовательностей метода `FillTrainingMat()`

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 colorOfCircuit(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;