1. **Разработка программных модулей**
   1. **Логика взаимодействия пользователя и графического интерфейса приложения**
      1. **Нажатие кнопки открытия папки с видеозаписями**

После того, как пользователь нажал на данный элемент графического интерфейса, происходят следующие действия:

1. Производится вызов диалогового окна, в котором пользователь производит выбор пути к директории, в которой находятся видеозаписи, которые необходимо обработать;
2. Происходит проверка пути на существование;
3. Если путь существует, производится создание объекта класса VideoFolder, иначе выводится соответствующее сообщение и переход к шагу номер 7;
4. Затем выполняется считывание всех путей к видеозаписям в выбранной директории;
5. Вычисляется количество найденных видеозаписей и их количество отображается на графической форме;
6. Выбранный путь сохраняется в настройки приложения;
7. Возврат из вызова функции.
   * 1. **Нажатие кнопки выбора директории для сохранения полученных после преобразования изображений**

После того, как пользователь нажал на данный элемент графического интерфейса, происходят следующие действия:

1. Производится вызов диалогового окна, в котором пользователь производит выбор пути к директории, в которой будут сохранены полученные узображения;
2. Происходит проверка пути на существование;
3. Если путь существует, производится сохранение данного пути в настройках приложения и отображения его в соответствующем элементе графической формы, иначе выводится соответствующее сообщение;
4. Возврат из вызова функции.
   * 1. **Нажатие кнопки преобразования видеозаписей**

После того, как пользователь нажал на данный элемент графического интерфейса, происходят следующие действия:

1. Проверяется выбранный путь к папке с видеозаписями;
2. Если путь пуст – выводится соответствующее сообщение и переход к шагу номер 11;
3. Если путь для сохранения полученных изображений пуст – выводится соответствующее сообщение и переход к шагу номер 11;
4. Пользователь оповещается о начале обработки и просьба подождать ее завершения;
5. Затем для каждого пути к видеозаписи из выбранной директории создается экземпляр класса FFMPEGConverter и производится вызов метода ConvertAll этого же класса;
6. После этого вычисляется количество папок для дальнейшего поиска дорожных знаков;
7. Затем это количество отображается на графической форме;
8. Возврат из вызова функции.
   1. **Преобразование видеозаписей в коллекцию изображений и координат**

Для преобразования базы видеозаписей к единому формату удобному для дальнейшей обработки используется класс FFMPEGConverter. Здесь происходит разбиения видеозаписи на кадры, получения субтитров, содержащих записи координат и времени видеосъемки.

* + 1. **Разбиение видеозаписи на кадры**

Для того чтобы преобразовать видеозапись в папку с изображениями используется функция ConvertVidToImages из класса FFMPEGConverter. Функция выполняет следующие шаги:

1. Создается переменная флага результата операции;
2. Если видеозапись по заданному существует выполняется шаг 3, если не существует – флаг результата устанавливается в нулевое значение;
3. Для каждой видеозаписи создается директория, в которую будут сохранены полученные изображения;
4. Высчитывается частота преобразования, равная отношению одной секунды в миллисекундах к числу отсчетов;
5. Создается переменная, содержащая параметры запуска утилиты ffmpeg, в которую записываются путь к видеозаписи, частота преобразования, путь к папке, в которую необходимо сохранить полученные изображения;
6. Создается и инициализируется переменная процесса для преобразования;
7. Устанавливаются параметры процесса: название запускаемого приложения(ffmpeg), аргументы запуска, режим с отображением консоли на экран для контролирования стадии преобразования;
8. Процесс запускается;
9. Программа ожидает завершения работы процесса;
10. Переменная флага результата устанавливается в единичное значение;
11. Возврат флага результата.
    * 1. **Получение субтитров из видеозаписи**

Для получения субтитров, содержащих географические координаты и время записи, используется функция ConvertVidToSubs из класса FFMPEGConverter. Функция выполняет следующие шаги:

1. Создается переменная, в которой хранится путь к текстовому файлу, в котором будут находится полученные субтитры;
2. Создается переменная, содержащая параметры запуска утилиты ffmpeg, в которую записываются путь к видеозаписи, частота преобразования, путь к папке, в которую необходимо сохранить полученные изображения;
3. Создается и инициализируется переменная процесса для преобразования;
4. Устанавливаются параметры процесса: название запускаемого приложения(ffmpeg), аргументы запуска, режим с отображением консоли на экран для контролирования стадии преобразования;
5. Процесс запускается;
6. Программа ожидает завершения работы процесса;
7. Возврат строки, содержащей путь к файлу с субтитрами.
   * 1. **Считывание файла с субтитрами и его преобразование к единому виду**

После получения субтитров из видеозаписи необходимо преобразовать записи из текстового файла к формату, пригодному для дальнейшей обработки. Для этого используется функция ParseSubtitleFile из класса FFMPEGConverter. Работа функции состоит из следующих шагов:

1. Происходит проверка текстового файла по введенному пути на предмет его существования.
2. Создается переменная строчного типа, в которую происходит считывание всего текстового файла;
3. Создается регулярное выражение для разбиения всей строки на отдельные записи, соответствующие каждому кадру видеозаписи;
4. С помощью регулярного выражения из строки получается коллекция записей;
5. Создается список, в котором будут находится географические координаты распознаваемых знаков;
6. Далее в цикле с помощью функции ParseMovementPoint этого же класса производится преобразование каждой полученной записи в объект класса MovementPoint, который хранит географические координаты;
7. Каждый полученный объект класса MovementPoint добавляется в список координат;
8. Возврат заполненного списка координат.
   * 1. **Функция создания объекта, содержащего географические координаты**

Для преобразования части строки, которая содержит географические координаты к объекту вызывается функция ParseMovementPoint. Данная функция работает следующим образом:

1. Сначала часть текстового файла разбивается на отдельные записи, которые в файле разделены между собой запятыми;
2. После этого для каждой записи производится преобразование даты в удобный для восприятия формат;
3. Затем производится преобразование географической широты в десятичный формат с помощью вызова функции ConvertDegreesAndDecimalMinutesStringToDecimal, логика работы которой описана ниже;
4. То же самое выполняется и для переменной, отражающей географическую долготу;
5. И в конце полученный результат преобразуется в объект класса movementPoint;
6. Возврат полученного объекта из функции.
   * 1. **Функция преобразования географических координат к десятичному формату**

Для того, чтобы привести координаты к единому десятичному формату, удобному для применения, используется функция ConvertDegreesAndDecimalMinutesStringToDecimal. Работает функция следующим образом:

1. Изначально координаты хранятся в дробном виде, разделенном точкой;
2. Вычисляем целые градусы;
3. Затем вычисляем целые минуты;
4. После этого приводим всё к целому количеству градусов, учитывая, что в одном градусе шестьдесят минут.
5. Возвращаем полученное значение.
   1. **Поиск** **контуров на изображении**

Для поиска контуров геометрических фигур на изображении используются методы класса ShapeDetection.

Метод detectShapes, позволяющий находить и выделять контуры геометрических фигур, таких как: треугольник, квадрат, прямоугольник, шестиугольник, круг, работает следующим образом:

Создается объект countors класса VectorOfVectorOfPoint, который будет хранить результаты поиска контуров;

1. Выполняется сглаживание введенного изображения с помощью фильтра Гаусса;

Затем вызывается метод FindCountors из библиотеки EmguCV, который выполняет поиск контуров геометрических фигур на изображении и возвращает объект класса VectorOfVectorOfPoint;

1. После этого для каждого найденного контура из массива countors выполняются следующие шаги;
2. Вычисляется периметр контура;
3. Производится аппроксимация контура для выявления прямоугольных границ найденной фигуры;
4. Затем рисуется рамка, ограничивающая найденную фигуру;
5. После этого вычисляется количество линий в контуре, для распознавания геометрической фигуры (3 грани – треугольник, 4 – четырехугольник, 6 – шестиугольник, и так далее).
6. Далее название найденной фигуры пишется на изображении;
7. И в конце производится возврат полученного изображения с нарисованными границами и названиями геометрических фигур.
   1. **Поиск** **определенного типа геометрических фигур на изображении**

Для данной цели предназначен методdetectShape класса ShapeDetection. Данный метод работает следующим образом:

1. Сначала полученное изображение переводится в оттенки серого цвета;
2. Затем выполняется сглаживание для уменьшения количества помех на исходном изображении;
3. Далее для поиска окружностей вызывается метод HoughCirlces из библиотеки EmguCV;
4. Результат поиска заносится в массив, содержащий объекты класса Circles;
5. После этого происходит преобразование исходного изображения с помощью вызова функции Canny, который работает на основе алгоритма Кэнни;
6. Затем для поиска линий вызывается метод HoughLinesP, результат работы которого заносится в массив lines класса LineSegment2D;
7. Далее для поиска треугольников и четырехугольников выполняется вызов функции FindCountors, результат работы которого заносится в созданный массив countors класса VectorOfVectorOfPoint;
8. Затем для каждого найденного контура выполняются следующие действия;
9. Производится аппроксимация контура для выявления прямоугольных границ найденной фигуры;
10. Затем необходимые контуры изображаются на исходном рисунке;
11. Метод возвращает полученное изображение, которое содержит контуры определенных геометрических фигур;
    1. **Поиск совпадений на нескольких изображениях**
12. Для поиска одного изображения на другом изображении написан класс ShapeComparation.Единственный метод FindMatch, который и производит поиск признаков одного изображения на другом работает так:
13. Создаются объекты класса VectorOfKeyPoint для каждого из изображений;
14. Создается экземпляр класса KAZE из библиотеки EmguCV, в котором находится программная реализация алгоритма поиска подобных областей на изображениях с помощью алгоритма KAZE;
15. Производится поиск ключевых точек на одном изображении и вычисляется его дескриптор;
16. Затем те же действия выполняются для другого изображения;
17. После этого создается объект класса DescriptorMatcher для поиска общих областей между изображениями;
18. C помощью этого объекта вызывается метод KnnMatch для поиска совпадающих областей;
19. Производится отсеивание повторяющихся совпадающих областей;
20. Производится отсеивание совпадающих областей, ориентация которых не совпадает с большинством других областей;
21. Выделяются найденные общие области;
22. Возврат из функции;
    1. **Поиск** **дорожных знаков на всех изображениях в выбранной папке с помощью каскадов Хаара**

После получения необходимых изображений и координат для проведения операции детекции используются методы класса DetectFolder и SignsHaarCascade.

Методы класса DetectFolder позволяют производить поиск дорожных знаков на совокупности фотографий, найденных в определенной директории.

Метод DetectAll класса DetectFolder работает следующим образом:

1. Создается коллекция, содержащая каскады, с помощью которых будет происходить поиск знаков;
2. Создается список, в котором будут находится пути ко всем папкам с изображениями, которые были найдены в введенной директории;
3. Происходит заполнение коллекции каскадов, путем создания объектов класса SignsHaarCascade;
4. В цикле каждое изображение приводится к единому формату Mat;
5. Для каждого каскада из списка вызывается метод DetectAll класса SignsHaarCascade, работа которого описана ниже;
6. При нахождении хотя бы одного дорожного знака, по введенному для сохранения полученных результатов пути производится создание директории с соответствующим названию исходного изображения названием;
7. Производится сохранение по заданному пути результатов поиска;
8. Возврат из функции.
   * 1. **Поиск** **дорожных знаков с помощью каскадов Хаара**

Поиск знаков на определенном изображении с помощью определенного каскада выполняется с помощью метода DetectAll.Работа метода DetectAll класса SignsHaarCascade происходит по следующему алгоритму:

1. Создается список, в котором будут находится части изображений, содержащих дорожные знаки;
2. С помощью каскада и вызова функции DetectMultiScale из библиотеки EmguCV производится поиск регионов на изображении, которые содержат дорожные знаки;
3. Для каждого найденного региона создается изображение и добавляется в список;
4. Функция возвращает полученный список.
   1. **Изменение размеров изображений и преобразование цветового пространства в градации серого**

Для того, чтобы изображения можно было классифицировать с помощью нейронной сети их необходимо привести единому виду:

* Размер 32 на 32 пикселя;
* Единственный цветовой канал (оттенки серого);
* Выровненное по контрасту.

Для данных целей используются следующие методы класса ImgOps.

* + 1. **Изменение размеров изображения**

Для того, чтобы размеры изображений были одинаковы и равны 32 на 32 пикселя используется метод InterpolationResize. Данный метод вызывает функцию Resize из библиотеки EmguCV. В нее передаются: само изображение, желаемый размер изображения после преобразования и выбирается метод интерполяции. В данном случае выбирается бикубическая интерполяция. Метод возвращает полученное изображение размером 32 на 32 пикселя.

* + 1. **Изменение цветового пространства изображения**

После изменения размеров изображения необходимо уменьшить число цветовых каналов для увеличения скорости классификации. Для этого используется метод RGBtoGrey, который из цветного изображения получает изображение в градациях серого. Данный метод работает так:

Сначала производится преобразование изображение в объект класса Image<Rgb, Byte>;

Затем вызывается метод Convert<Gray, Byte>, который выполняет непосредственно само преобразование;

1. Метод возвращает полученное изображение в градациях серого.
   * 1. **Контрастное выравнивание изображения**

Последней стадией подготовки к классификации изображения является контрастное выравнивание. Оно необходимо для того, чтобы избежать появления неосвещенных и на оборот слишком освещенных областей на изображении, что могло бы привести к ухудшению качественных характеристик классификации нейронной сети. Для этого используется метод ContrastAlignment. Данный метод вызывает функцию CLAHE из библиотеки EmguCV. В нее передаются: само изображение, фактор контрастности, который предотвращает перенасыщенность изображения, особенно в однородных областях, а также размер окна фильтра. Метод возвращает полученное изображение.

* 1. **Нейронная сеть**

Классификация подготовленных изображений происходит с помощью сверточной нейронной сети, программная реализация которой находится в классе CNN.

* + 1. **Создание нейронной сети**

Создание экземпляра программной реализации нейронной сети выполняется с помощью вызова метода CreateCNN класса CNN. Этот метод работает так:

1. Производится начальная инициализация параметров нейронной сети, а именно: точность, количество слоев, флаг обученности нейронной сети и путь к сохранению параметров сети;
2. Затем производится добавление слоев нейронной сети: входного слоя, сверточных слоев, слоев пулинга, активации, полносвязных слоев, слоя softmax c соответствующими параметрами для каждого из этих слоев;
3. После этого сохраняется количество классов распознаваемых дорожных знаков;
4. Возврат из функции.
   * 1. **Обучение нейронной сети**

Обучение сверточной нейронной сети (вычисление подходящих весовых коэффициентов для нейронов) производится с помощью метода TeachCNN класса CNN. Этот метод принимает пути к обучающей и тренировочной выборкам, ожидаемую точность, скорость обучения, размер пачки изображений для обучения. В данном методе выполняются следующие шаги:

1. Производится проверка количества слоев: если количество слоев равно нулю, производится инициализация сети;
2. Выполняется создание обучающего и тестового наборов изображений;

Производится создание экземпляра класса SgdTrainer<double>, который является «учителем» для нейронной сети, который производит обучение методом градиентного спуска;

1. Задаются параметры «учителя», такие как скорость обучения и размер пачки изображений для обучения;
2. До тех пор, пока достигнутая точность меньше необходимой выполняются следующие действия;
3. Создается пачка из обучающей выборки с заранее заданным размером;
4. Каждое изображение пропускается через сеть для проверки и корректировки весовых коэффициентов;
5. Затем создается пачка из тестовой выборки с заранее заданным размером;
6. Каждое изображение пропускается через сеть для тестирования и вычисления достигнутой точности распознавания;
7. Выводятся достигнутая точность и номер текущей итерации обучения;
8. Конец цикла;
9. Флаг обученности сети выставляется в единичное состояние;
10. Возврат из функции.
    * 1. **Сохранение данных нейронной сети**

Сохранение данных нейронной сети позволяет избежать постоянного проведения обучения с помощью сохранения результатов в текстовый файл с возможностью последующей их загрузки. Сохранение данных выполняется с помощью вызова метода SaveCNN. Данный метод работает следующим образом:

1. Производится проверка количества слоев: если количество слоев равно нулю, выполняется возврат из функции;
2. Производится проверка флага обученности нейронной сети: если количество слоев равно нулю, выполняется возврат из функции;
3. Выполняется преобразование параметров обученной сети к строковому формату пригодному для сохранения результатов;
4. Выполняется создание переменной, в которой хранится название файла, в которое включены ключевые параметры нейронной сети;
5. Если файл с таким названием существует, производится его удаление;
6. Выполняется создание нового текстового файла с заранее определенным названием;
7. Производится запись данных нейронной сети в текстовый файл;
8. Возврат из вызова функции.
   * 1. **Загрузка данных нейронной сети**

Загрузка данных ранее обученной нейронной сети выполняется с помощью вызова функции LoadCNN. Данный метод работает следующим образом:

1. Сначала производится проверка существования текстового файла пути загрузки;
2. Затем происходит считывание всего текстового файла в строковую переменную;
3. После этого выполняется попытка преобразования строки к виду экземпляра нейронной сети;
4. Если это не удалось, выполняется возврат из вызова функции с результатом равным минус одному, иначе выполняется шаг под номером 5;
5. Производится замена текущего экземпляра класса, загруженным из файла;
6. Метод возвращает количество слоев в загруженной нейронной сети;
   * 1. **Загрузка выборок для обучения нейронной сети**

Загрузка выборок осуществляется с помощью вызова функции Load класса DataSets. Работа функции заключается в следующем:

1. Выполняется считывание по заданному пути папки с тренировочными изображениями, используя метод Load класса ImageReader, логика работы которого описана ниже;
2. Выполняется считывание по заданному пути папки с тестовыми изображениями и файлом номеров классов, соответствующих этим изображениям, используя метод Load класса ImageReader, логика работы которого описана ниже;
3. Производится проверка количества загруженных изображений;
4. Если не загружено ни одного изображения, то производится возврат из вызова данной функции с результатом false, иначе выполняется шаг под номером 5;
5. Выполняется создание тренировочного набора данных из загруженной обучающей выборки;
6. Выполняется создание тестового набора данных из загруженной тестовой выборки;
7. Производится возврат из функции с результатом true;
   * 1. **Загрузка выборки из папки по определенному пути**

Загрузка каждой выборки состоит из загрузки изображений, номеров классов этих изображений и объединение изображения с номером соответствующего ему класса. При каждой загрузке обучающей выборки происходит обновление файла с номерами классов. Это делается для возможности пополнения обучающей базы новыми изображениями и новыми классами изображений. Загрузка изображений производится по этому алгоритму:

1. Создается список, который будет содержать изображения в формате массива байт;
2. Выполняется создание экземпляра класса ImageFolder по введенному пути;
3. Каждое найденное изображение преобразуется в массив байт и добавляется в список;
4. Функция возвращает полученный список;

Обновление файла с номерами с последующей их загрузкой выполняется следующим образом:

1. Создается список, который будет содержать номера классов в целочисленном формате;
2. Проверяется наличие файла с таким же названием и если он существует – производится его удаление;
3. Создается новый файл для сохранения новых номеров классов изображений;
4. Для каждого изображения считывается его название;
5. Выбирается часть названия, которая включает в себя номер класса, к которому принадлежит изображение;
6. Полученный номер записывается в файл и добавляется в список;
7. Функция возвращает заполненный список;
   1. **Сохранение полученных результатов**

Каждый найденный на изображении дорожный знак трансформируется в объект класса Result для дальнейшего сохранения на локальном хранилище и дальнейшего экспорта в базу данных.

Результаты работы программы являются массивом элементов вида Result.

Для сохранения результатов на локальном хранилище используется функция ExportToFile класса ResultExport. Данный метод записывает результаты в файл формата xml по введенному пути с помощью методов класса XmlSerializer. Запись результатов происходит следующим образом:

1. Создается экземпляр класса XmlSerializer;
2. Комбинируется путь к папке для сохранения результатов и имени файла;
3. С помощью потока записи в файл FileStream и метода Serialize класса XmlSerializer производится запись массива результатов;
4. Возврат из вызова функции.
   1. **Экспорт результатов в базу данных**

Экспорт результатов в базу данных происходит после того, как результаты были сохранены на локальном хранилище. Для этого используются методы класса ResultExport.

* + 1. **Экспортирование папки с результатами**

Сначала вызывается метод ExportFolder, который позволяет проводить экспорт всех файлов формата xml, которые были сохранены ранее в выбранной директории. В него передается путь к директории. Все происходит следующим образом:

1. Сначала обновляется объект xmlFolder класса XmlFolder, в котором хранятся пути к файлам результатов;
2. Затем пути файлов загружаются в массив files;
3. После этого вызывается метод ExportToDB этого же класса, логика работы которого описана ниже;
4. Выполняется возврат из вызова функции.
   * 1. **Экспортирование файла с результатами**

Экспортирование файла формата xml c результатами работы программы происходит с помощью вызова метода ExportToDB. Данный метод выполняет следующие действия:

1. Для считывания массива результатов из файла вызывается метод UploadFromFile этого же класса, логика работы которого описана ниже;

Затем создается объект dt класса DataTable, который является промежуточным звеном в цепочке экспортирования;

После этого добавляются все нужные столбцы таблицы dt в базе данных, которые соответствуют полям класса Result;

Затем каждый элемент массива добавляется в соответствующую строчку таблицы dt;

1. Далее производится попытка подключения к удаленной базе данных с помощью введенных параметров;
2. При успешном подключении таблица приводится к пригодному для экспортирования виду;
3. Далее с помощью консоли производится вызов программы для соединения с базой данных и занесение результатов;
4. В конце выполняется закрытие соединения;
5. Соединение с базой данных закрывается;
6. Возврат из вызова функции.