1. **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Рассмотрим подробно функционирование программы. Для этого проведем анализ основных блоков программы и рассмотрим их зависимости. А также проанализируем все функциональные компоненты, которые входят в состав кода программы, и рассмотрим назначение всех методов и переменных классов этих блоков.

В разрабатываемом приложении можно выделить следующие блоки:

* блок пользовательского интерфейса;
* блок разделения видеозаписей на кадры;
* блок обработки изображения;
* блок детектирования;
* блок стандартизации изображений;
* блок классификации;
* блок приведения полученных данных к одному формату;
* блок экспортирования результатов в базу данных.

Изначально пользователь попадает на главный экран, логика которого находится в классе Form1. Здесь расположен весь основной графический интерфейс. В данном классе находятся все элементы, через которые пользователь взаимодействует с приложением. Здесь отображаются все информационные сообщения и выводятся управляющие элементы. Пользователь управляет программой в основном через нажатие на клавиши, расположенные в окне приложения. При нажатии происходит вызов необходимых функций, происходят определенные операции и пользователь получает какое-либо уведомление о завершении произошедшей операции.

Главный экран приложения разделен на составные блоки, в соответствии с этапом обработки входных данных.

На главном экране пользователь может указать на папку, в которой находятся видеозаписи, на которых необходимо произвести распознавание. После этого производится выбор места расположения для полученных изображений, которые будут сохранены во вложенные папки в соответствии с названиями исходных видеозаписей. Также при обработке исходных видеоматериалов есть возможность настройки частоты создания изображений. После этого пользователь выбирает набор необходимых преобразований и фильтров для улучшения качества распознавания. Затем он может выполнить операцию детекции областей изображений, на которых есть дорожные знаки. После этого производится запуск классификатора обнаруженных знаков. И в конце происходит запись результатов в базу данных с заранее определенными полями.

* 1. **Классы разрабатываемого программного средства**
     1. **Класс ffmpegConverter**

Этот класс является функциональной оберткой консольного вызова программы ffmpeg из одноименного набора библиотек для разбиения с определенной частотой видеозаписи на изображения.

Поле sourcePath хранит в себе путь к набору видеозаписей для распознавания.

В поле fps хранится частота преобразования видеозаписи, то есть количество кадров за секунду видеозаписи.

Метод convertVidToImages() создает новый процесс, в котором через консоль Windows вызывает программу ffmpeg c определенным набором аргументов. Через аргументы передается расположение полный путь к видеозаписи и папки для сохранения полученных изображений, а также количество получаемых кадров за секунду.

Аргумент -qscale указывается для того, чтобы использовать фиксированную шкалу качества. Аргумент –r устанавливает частоту кадров. Пример команды вызова программы ffmpeg из консоли ОС Windows приведен на рисунке 3.1.



Рисунок 3.1 – Пример команды ffmpeg для раскадрирования видеозаписи

Метод convertVidToSubs создает новый процесс, в котором через консоль Windows вызывает программу ffmpeg c определенным набором аргументов. Через аргументы передается расположение видеозаписи, некоторые дополнительные параметры и путь к новому файлу для сохранения полученных изображений.

Аргументы -vn и -an используются, чтобы пропустить включение потоков видео и аудио соответственно, независимо от того, отображены ли они вручную или автоматически, за исключением тех потоков, которые являются выходами сложных фильтров.

Аргумент -map нужна для ручного управления выбором потока в каждом выходном файле.

Пример команды вызова программы ffmpeg из консоли ОС Windows приведен на рисунке 3.2.

Метод convertAll() вызывает два предыдущих метода для полного преобразования набора видеозаписей в соответствующее число папок, содержащих изображения в формате .jpg и файлы субтитров в формате .txt. Он в зависимости от успешности выполнения включенных методов возвращает либо *true,* либо *false.*



Рисунок 3.2 – Пример команды ffmpeg для получения субтитров из видеозаписи

* + 1. **Класс ImgOps**

Данный класс содержит методы для преобразования изображений с целью увеличения количественно-качественных характеристик распознавания дорожных знаков. Он использует методы библиотеки EmguCV.

Используя метод *RGBtoGrey,* происходит преобразование изображения из цветного в градации серого. Это позволяет уменьшить количество цветовых каналов с 3 до 1, что позволяет использовать меньше вычислительных мощностей и уменьшает время распознавания. Он принимает изображение в цветовом пространстве RGB и возвращает изображение в градациях серого цвета.

Метод *RGBtoHSV* преобразует изображение из цветового пространства RGB в цветовое пространство HSV. Он принимает изображение в цветовом пространстве RGB и возвращает изображение в цветовом пространстве HSV.

Для того чтобы получить бинарное изображение используется метод *toBinary*. Он принимает изображение, а также порог бинаризации. На выходе получается изображение, в котором каждый пиксель принимает значение “1” либо “0”. Если яркость пикселя исходного изображения меньше порога бинаризации, то в преобразованном изображении его значение будет равно нулю, если же больше – единице.

В методе *InterpolationResize* производится масштабирование изображения с использованием бинарной интерполяции. Данный метод на входе получает исходное изображение и необходимые размеры результата и возвращает полученное изображение.

Суть интерполяции заключается в использовании имеющихся данных для получения ожидаемых значений в неизвестных точках. Интерполяция изображений работает в двух измерениях и пытается достичь наилучшего приближения в цвете и яркости пикселя, основываясь на значениях окружающих пикселей. Бикубическая интерполяция рассматривает массив из 4x4 окружающих пикселей – всего 16. Поскольку они находятся на разных расстояниях от неизвестного пикселя, ближайшие пиксели получают при расчете больший вес. Бикубическая интерполяция производит значительно более резкие изображения, чем другие методы, и возможно, является оптимальной по соотношению времени обработки и качества на выходе. Результат увеличения с интерполяцией можно увидеть на рисунке 3.2(а).





а – исходное изображение; б – после увеличения с интерполяцией

Рисунок 3.2 Результат увеличения с интерполяцией



Рисунок 3.3 Результат увеличения без интерполяции

Для улучшения различимости элементов классифицируемого фрагмента изображения содержащего дорожный знак необходимо применить метод *ContrastAlignment*. Метод принимает исходное изображение и возвращает масштабированное изображение. В данном методе используется контрастное выравнивание CLAHE (Contrast-limited adaptive histogram equalization).

Контрастное выравнивание CLAHE используется для изображений, имеющих неоднородное геометрическое распределения яркостей. Оно анализирует небольшие участки изображения и позволяет усилить локальный контраст. Для каждого пикселя рассматривается небольшая окрестность изображения, по которой строится функция преобразования, при этом все изображение, как таковое, не используется. Оно позволяет уменьшить неоднородность освещения дорожных знаков. Результат контрастного выравнивания можно увидеть на рисунке 3.4.



а – исходное изображение; б – после контрастного выравнивания

Рисунок 3.4 Результат контрастного выравнивания CLAHE

В методе *Filter* происходит фильтрация изображения, после которой на готовом изображении будут находиться только те области изображения, цвет которых находится в заданном HSV интервале. Метод принимает нижний и верхний порог фильтрации и возвращает полученное изображение.

* + 1. **Класс SignsHaarCascade**

Этот класс выполняет распознавание знаков на основе выбранных каскадов Хаара. Класс содержит экземпляр класса CascadeClassifier из библиотеки EmguCV.

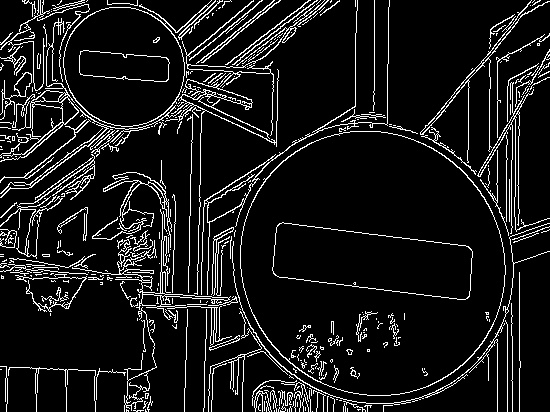
При создании объекта этого класса выбирается существующий на данном компьютере заранее обученный каскад в файле формата XML, который будет использоваться для детекции знаков в базе фотографий.

Метод *detectAll* принимает изображение, на котором необходимо найти знаки. Здесь вызывается функция *DetectMultiScale* класса *CascadeClassifier* из библиотеки EmguCV, который находит прямоугольные области в изображении, которые, вероятно, содержат объекты, для которых обучен каскад, и возвращает эти области в виде последовательности прямоугольников. Функция сканирует изображение несколько раз и в разных масштабах. Каждый раз он учитывает перекрывающиеся области на изображении. Также может быть применена некоторая эвристика для уменьшения количества анализируемых областей, например, алгоритм Кэнни.

Детектор границ Кэнни - оператор [обнаружения границ](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86) изображения. Границы здесь отмечаются там, где градиент изображения приобретает максимальное значение. Они могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения горизонтальных, вертикальных и диагональных ребер в предварительно размытом для удаления шумов изображении. Результат работы данного детектора можно увидеть на рисунке 3.5.

Далее вызывается метод *getListOfROI,* который принимает коллекцию регионов интереса– областей изображения, в которых были найдены нужные знаки. Метод возвращает коллекцию, содержащую части исходного изображения в ранее обнаруженных регионах интереса.





а – исходное изображение; б – после применения детектора

Рисунок 3.4 – Результат применения детектора границ Кэнни

* + 1. **Класс OpenHaarCascadeFileDialog**

Данный класс предназначен для создания окна, в котором пользователь выбирает файл формата XML, в котором хранится результат обучения каскада Хаара.

Главным методом этого класса является *openCascade*. При его вызове создается стандартное окно указания пути ОС Windows, с помощью которого пользователь указывает путь к нужному каскаду Хаара. Метод возвращает строку, в которой находится полный путь к выбранному каскаду.

* + 1. **Класс OpenVideoFolderFileDialog**

Этот класс нужен для вывода на экран окна для выбора папки с видеозаписями, которые нужно использовать для пополнения базы данных новыми данными.

Главным методом этого класса является *openFolder*. При его вызове создается стандартное окно указания пути ОС Windows, с помощью которого пользователь указывает путь к папке. Из нее программа получает доступ к видеозаписям, которые необходимо обработать и выдать нужный результат. Метод возвращает массив строк, в котором находится полный путь к видеозаписям из выбранной папки.

* + 1. **Класс OpenPictureFolderFileDialog**

Этот класс используется для вывода на экран окна, которое используется для выбора папки с изображениями, которые нужно использовать для нахождения знаков.

Главным методом этого класса является *openFolder*. При его вызове создается стандартное окно указания пути ОС Windows, с помощью которого пользователь указывает путь к папке, из которой программа может получить доступ к изображениям. Метод возвращает массив строк, в котором находится полный путь к изображениям из выбранной папки.

* + 1. **Класс ImageFolder**

Этот класс используется для создания объектов, которые содержат путь к папке с фотографиями. Также здесь хранятся пути к данным фотографиям и их количество в определенной папке.

Поле *path* хранит полный путь к папке с изображеними.

Поле *count* показывает количество изображений, которые находятся в папке по адресу path.

* + 1. **Класс ShapeDetection**

В этом классе находятся функции поиска геометрических фигур на изображении.

Метод *detectShapes* принимает изображение, производит поиск геометрических фигур (треугольники, квадраты, прямоугольники, круги, шестиугольники), выделяет их и возвращает полученное изображение. Поиск производится с помощью метода контурной аппроксимации.

В методе detectShape производится поиск определенных фигур. При вызове функции указывается, какие фигуры нужно обнаружить, очертить и вернуть полученное изображение.

* + 1. **Класс VideoFolder**

Этот класс используется для создания объектов, которые содержат путь к папке с видеозаписями. Также здесь хранятся и сами видеозаписи, и их количество в определенной папке.

Поле *path* хранит путь к набору видеозаписей.

Поле *count* показывает количество видеозаписей, которые находятся в пути path.

* 1. **Сверточная нейронная сеть**
     1. **Класс Neuron**

В данном классе реализуется наименьшая единица нейронной сети – нейрон.

Поле *type* формата NeuronType хранит тип нейрона из файла с перечислениями (Enums).

В массиве типа double[] *weights* хранятся веса соответствующего нейрона.

В массиве типа double[] *inputs* содержатся все поступающие на него значения.

Поле формата double *output* хранит значение выхода нейрона.

В поле *a* находится угловой коэффициент нейронов для использования в функции активации вида RelU с “утечкой” (LeakyReLU).

Метод *Activator* выполняет нелинейные преобразования для нейронов слоя свёртки, полносвязного слоя и выходного слоя.

Метод *LeakyReLU* содержит реализацию алгоритма LeakyReLU. При получении положительного значения возвращается единица, а при получении отрицательного – значение поля *a.*

* + 1. **Класс InputLayer**

В этом классе содержится функционал входного слоя сети. В зависимости от текущей выборки (тренировочной либо тестовой) считывает пиксели изображений с каждого изображения соответствующей выборки для передачи их значений следующим слоям. Входной слой учитывает двумерную топологию изображений и состоит из нескольких матриц, где каждая карта соответствует изображению с конкретным каналом (красным, синим и зеленым). В силу того, что планируется обрабатывать изображения в размере 32\*32 пикселя и изображения находятся в градации серого цвета, на данном слое находится 1024 нейрона.

* + 1. **Класс Layer**

Класс Layer является абстрактным, который не содержит общие свойства и методы слоёв нейронной сети, так как слои очень похожи друг на друга, но имеют различия в логике работы. Сам по себе в работе сети данный класс не учувствует, участвуют только его наследники. Каждый слой содержит определенное число нейронов соответствующего вида и веса для связи с нейронами предыдущего слоя. Нейроны объединяются в массивы.

Поле *neuronsCount* отображает количество нейронов на слое.

Поле *neuronsprevCount* отображает количество нейронов на предыдущем слое.

Константа *learningRate* отображает скорость обучения определенного слоя сети. Скорость обучения нужна для настройки коэффициента изменения весов нейронов.

Константа *momentum* отображает момент инерции определенного слоя сети. Она так же участвует в изменения весов нейронов.

В массиве типа double[] *lastdeltaweights* содержатся веса предыдущей итерации обучения.

Массив *Neurons*[] является непосредственной коллекцией нейронов для данного слоя сети.

В методе *WeightInitialize* происходит синхронизация весов нейронов слоя и их значений в файле XML. Здесь происходит записи весов в файл при обучении сети и чтение весов во время операции распознавания. После каждого изменения весов происходит обновления файла и его сохранение.

Абстрактный метод *Recognize* определяет формат функции распознавания для классов-наследников.

Абстрактный метод *BackwardPass* определяет формат функции обучения для классов-наследников.

* + 1. **Класс ConvolutionalLayer**

Данный класс реализует функции сверточного слоя сети и наследует свойства и методы абстрактного класса Layer, но имеет собственную логику, соответствующую специфике решаемых задач этого слоя. Сверточный слой представляет собой набор карт признаков, у каждой карты есть синаптическое ядро (фильтр).

В массиве типа double[] *kernel* хранится ядро свертки. Ядро представляет собой фильтр или окно, которое скользит по всей области предыдущей карты и находит определенные признаки объектов. Размер ядра в данном случае составляет 5\*5. Данный объект является одной из главных особенностей сверточной нейросети. В обычной многослойной сети очень много связей между нейронами, то есть синапсов, что весьма замедляет процесс детектирования. В сверточной сети же – наоборот, общие веса позволяет сократить число связей и позволить находить один и тот же признак по всей области изображения.

В массивах типа double[] *output0 – output7* хранятся полученные карты сверточного слоя.

В методе *CalculateError* происходит расчет ошибки на данном слое. Метод возвращает величину ошибки в формате double.

* + 1. **Класс SubsamplingLayer**

В этом классе находится реализация слоя субдискретизации нейронной сети. Данный слой следует после сверточного слоя. Он, как и сверточный имеет карты, количество которых совпадает с предыдущим (сверточным) слоем. Цель слоя – уменьшение размерности карт предыдущего слоя. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. В процессе сканирования ядром слоя фильтром карты предыдущего слоя, сканирующее ядро не пересекается в отличие от сверточного слоя. Здесь имеет место операция пулинга (pooling). Операция Pooling – это усреднение либо взятие максимума. Она работает на окнах размером 2\*2. Накладывая на изображение окно, выбирается максимальный элемент из этой области и отправляется на выход. Таким образом, происходит уменьшение размера изображение. Это дает небольшую инвариантность к смещениям. Данный слой позволяет нейросети быть более устойчивой к сдвигам изображения.

* + 1. **Класс FullyconnectedLayer**

Здесь реализуется функционал полносвязного слоя сети и наследует свойства и методы абстрактного класса Layer, но имеет особую логику, соответствующую специфике решаемых задач этого слоя. На данном слое каждый нейрон соединен со всеми нейронами на предыдущем уровне, причем каждая связь имеет свой весовой коэффициент.  Цель полносвязного слоя – классификация, моделирование сложной нелинейной функции, оптимизируя которую, улучшается качество распознавания. Каждый нейрон в этом слое – персептрон с нелинейной функцией активации.

* + 1. **Класс OutputLayer**

Данный класс реализует функции выходного слоя сети и наследует свойства и методы абстрактного класса Layer, но имеет собственную логику, соответствующую специфике решаемых задач этого слоя. Выходной слой связан со всеми нейронами предыдущего слоя. Количество нейронов соответствует количеству распознаваемых классов.

* + 1. **Класс SoftmaxLayer**

Здесь описаны функции дополнительного слоя “Softmax”. Данный слой добавлен для интерпретации выхода сети. Значения данного слоя находятся в интервале от нуля до единицы и их можно рассматривать как вероятность принадлежности найденного знака к тому либо иному классу. В данном классе производится сопоставление номера класса и названия дорожного знака для отображения, чтобы отобразить полученные результаты в удобном для пользователя виде.

Поле данных формата Dictionary *pairs* содержит номера классов знаков и их названия в формате ключ-значение.

* + 1. **Класс CNN**

Данный класс реализует непосредственно саму сверточную нейронную сеть. Он содержит все слои входящие в нейронную сеть. Этот класс является сборщиком всей сети, так как скрепляет все слои воедино и занимается её обучением и тестированием.

Массив данных формата double *Results* содержит результаты работы всей сети. Количество элементов массива соответствует количеству классов знаков. Каждый элемент принимает значение от нуля до единицы, что соответствует вероятности принадлежности распознавания знака к определенному классу.

В методе *Train* производится обучение сети. На начальном этапе нейронная сеть является необученной (ненастроенной). Под обучением понимается последовательное предъявление образов на вход нейросети, из обучающего набора, затем полученный ответ сравнивается с желаемым выходом, полученная разница между ожидаемым ответом и полученным является результат функции ошибки (дельта ошибки). Затем эту дельту ошибки необходимо распространить на все связанные нейроны сети. Таким образом, обучение нейронной сети сводится к минимизации функции ошибки, путем корректировки весовых коэффициентов синаптических связей между нейронами. Обучение происходит по эпохам. После изменения весов происходит изменение их значений в файле конфигурации. Для обучения описанной нейронной сети был использован алгоритм обратного распространения ошибки.

В методе *ForwardPass* происходит передача данных между слоями. Этот метод принимает значения выходного слоя и передает их по заранее определенному порядку слоев.

В методе *Recognize* происходит непосредственно распознавание дорожного знака на основе обучающей выборки.

* + 1. **Файл Enums**

В этот файл отдельно вынесены используемые перечисления. Среди них:

* XMLAccessMode, в котором содержатся виды операций для взаимодействия с XML файлом, в котором находятся веса нейросети. Значение Get соответствует операции операции чтения, Set – операции записи.
* NeuronType, где перечисляются типы вычислений нейрона для соответствующего слоя. Значение Convolutional соответствует нейрону сверточного слоя, Fullyconnected – полносвязного, Subsampling – слоя субдискретизации, Output – выходного слоя.
* NetworkMode описывает режимы работы сети. Значение Train говорит о том, что сеть находится в режиме обучения, Test – тестирования, Work – непосредственно работы.
  1. **Структура организации Properties.Settings**

Организация Properties.Settings — это xml файл, который можно найти в папке пользователя. Данный файл позволяет хранить и получать доступ к значениям, которые сохраняются между сеансами выполнения приложения. Эти значения называются *параметры*. Используя параметры, могут быть записаны пользовательские настройки или ценные сведения, которые приложению необходимо использовать.

Планируется хранить следующие настройки приложения:

* номер версии приложения;
* последний выбранный путь к папке с видеозаписями;
* последний выбранный путь к папке с изображениями;
* последний выбранный путь к обученному каскаду;
* путь к файлу формата XML, в котором хранятся веса нейронов для сверточной нейронной сети.
* флаг первого открытия приложения для отображения сообщения с инструкциями по пользованию.
  1. **Экспорт результатов в базу данных**
     1. **Класс ResultExport**

В этом классе происходит связь с базой данных и запись новых результатов.

Поле *login* содержит имя пользователя для подключения к базе данных.

Поле *password* содержит пароль для подключения.

Поле *dataSource* указывает значение, соответствующее атрибуту источника данных.

В методе *ConnectToDB* происходит подключение к базе данных с помощью связки логина и пароля и проверяется доступность базы данных. Метод возвращает true, если соединение установлено и база данных доступна, либо false, если произошла ошибка.

Метод *Export* производит создание в базе данных новой записи, содержащей данные обнаруженного знака.

Метод *Find* нужен для предотвращения повторной записи дорожного знака, если он был обнаружен на другой видеозаписи. Поиск происходит по географическим координатам, а также номеру и километру дороги. Если хотя бы одна пара значений (долгота и широта, либо номер дороги и километр) совпали, то происходит лишь обновление даты.

Метод *CloseConnection* производит закрытие соединения с базой данных.

* + 1. **Класс Result**

Этот класс служит для стандартизации результатов обработки и распознавания.

Поле *id* отображает уникальный номер знака.

Поле *signClass* содержит номер класса, к которому принадлежит знак.

Поле *lattitude* показывает географическую широту.

Поле *longitude* показывает географическую долготу.

Поле *date* содержит время на видеозаписи, когда дорожный знак был заснят.

Поле *roadId* указывает Id дороги, на которой расположен знак

Поле*roadKm* показывает километр от начала дороги, на которой расположен знак.

* 1. **Хранение полученных результатов в базе данных Oracle**

Для сохранения результатов в базе данных Oracle будет использоваться таблица с заранее определенными полями. Их можно увидеть в таблице 3.1.

Таблица 3.1 Структура таблицы в базе данных

|  |  |
| --- | --- |
| 1 Id | Уникальный номер знака |
| 2 Class | Вид знака |
| 3 Lattitude | Широта |
| 4 Longitude | Долгота |
| 5 Date | Дата обнаружения знака на видеозаписи |
| 6 RoadId | Id дороги, на которой расположен знак |
| 7 RoadKm | Километр от начала дороги, на которой расположен знак |