4. Проектирование и разработка алгоритмов и структур данных

4.1. Структурные и функциональные модели

4.1.1. Анализ информационных потоков для разработки ПО

Функциональная модель представляет собой набор диаграмм потоков данных (далее - ДПД), которые описывают смысл операций и ограничений. ДПД отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние хранилища данных. ДПД - это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.

ДПД содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.

Процесс преобразует значения данных. Процессы самого нижнего уровня представляют собой функции без побочных эффектов (примерами таких функций являются вычисление суммы двух чисел, вычисление комиссионного сбора за выполнение проводки с помощью банковской карточки и т.п.). Весь граф потока данных тоже представляет собой процесс (высокого уровня). Процесс может иметь побочные эффекты, если он содержит нефункциональные компоненты, такие как хранилища данных или внешние объекты.

Поток данных соединяет выход объекта (или процесса) со входом другого объекта (или процесса). Он представляет промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных; примеры стрелок, изображающих потоки данных.

Активным называется объект, который обеспечивает движение данных, поставляя или потребляя их. Активные объекты обычно бывают присоединены к входам и выходам ДПД. На ДПД активные объекты обозначаются прямоугольниками.

Хранилище данных - это пассивный объект в составе ДПД, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам.

ДПД показывает все пути вычисления значений, но не показывает в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение в функциональную модель необязательно. Тем не менее иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса.

Рассмотрим подробнее этап проектирования программы. Для этого опишем общую структуру системы, представленную на рисунке 4.1.

На данной диаграмме присутствуют следующие потоки:

- Принадлежность лица пользователю;
- Расположение лица;
- Изображение лица;
- Параметры ГА.

Диаграмма потоков для системы отображения результатов представленная

на рисунке 4.2.

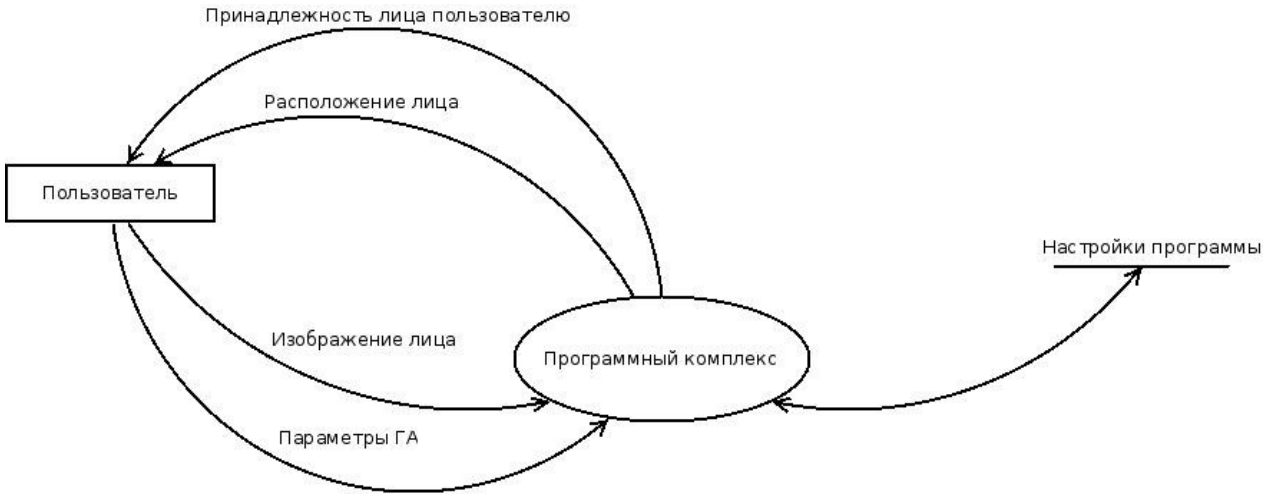

Рисунок 4.1 - Общая структура системы

Рисунок 4.2 - Подсистема отображения результатов

На данной диаграмме присутствуют следующие потоки:

- Изображение с камеры;
- Выделенная область лица — прямоугольник который показывает границы найденного лица;
- Решение о принадлежности лица — информация о том кому принадлежит найденное лицо.

Подсистема работы с генетическими алгоритмами представленная на

рисунке 4.3.

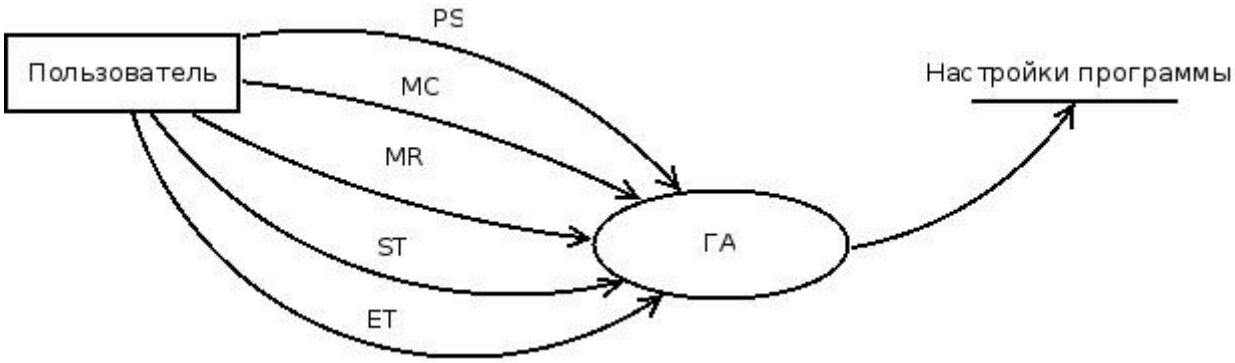

Рисунок 4.3 - Подсистема работы с ГА

На данной диаграмме присутствуют следующие потоки:

- PS - размер популяции;
- MC - начальная вероятность мутации;
- MR - начальная величина мутации;
- ST - начальная температура имитации отжига;
- PS - конечная температура имитации отжига.

4.1.2. Структура программного комплекса

Общая схема разрабатываемого программного обеспечения представленная на рисунке 4.4.

Рассмотрим каждую из подсистем подробнее:

Пользователь — В данном случае это не под система. Так на диаграмме представлена схема работы с пользователем программы.

Интерфейс программы — Данная подсистема отвечает за отрисовку интерфейса пользователя, сбор изображений лиц, отображения результата распознавания.

Обучение — Отвечает за обучение нейронной сети на обнаружение и распознавание лица.

ГА — отвечает за работу генетических алгоритмов.

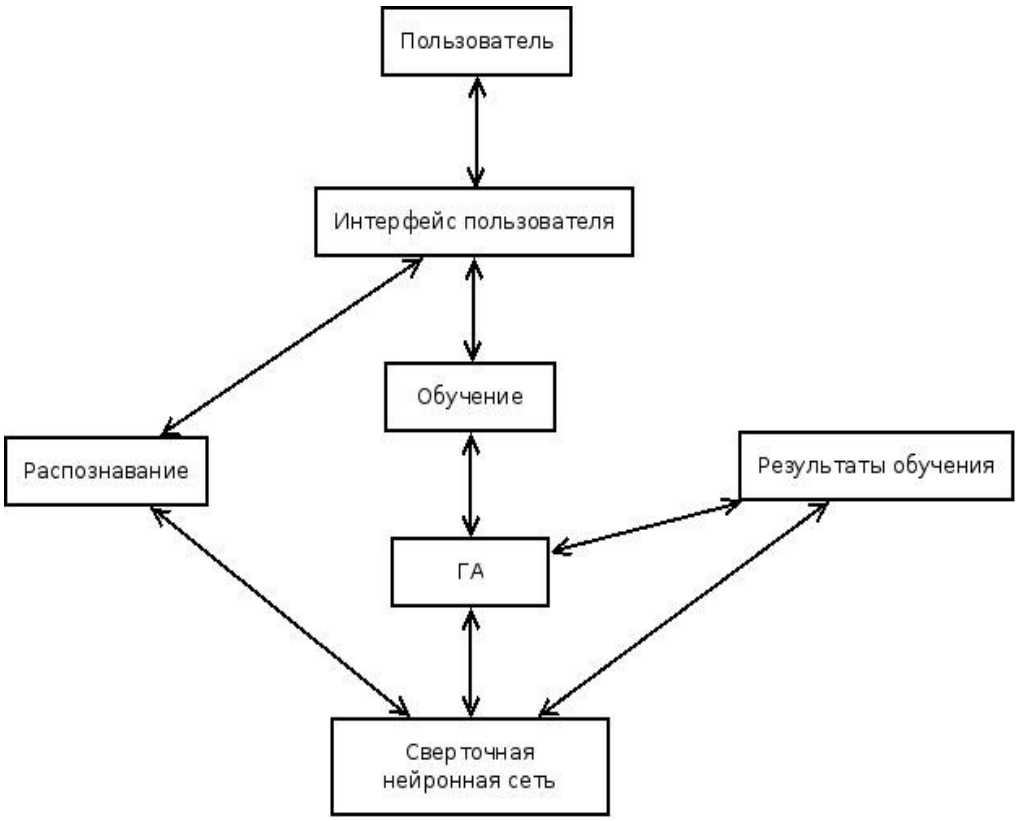

Рисунок 4.4 - Общая структура системы

Сверточная нейронная сеть — Данная подсистема отвечает за работу сверточной нейронной сети.

Распознавание — отвечает за обнаружение и распознавание лиц.

Результаты обучения — отвечает за хранение и работу с данными, полученными в результате процесса обучения.

4.2. Проектирование диаграммы классов

UML (сокр. от англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML моделью. UML был создан для определения, визуализации, проектирования и документирования в основном программных систем. UML не является языком программирования, но в средствах выполнения UML-моделей как интерпретируемого кода возможна кодогенерация.

Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (generalization), объединение (aggregation) и поведение) и больше сконцентрироваться на проектировании и архитектуре.

В 1994 году Гради Буч и Джеймс Рамбо, работавшие в компании Rational Software, объединили свои усилия для создания нового языка объектно-ориентированного моделирования. За основу языка ими были взяты методы моделирования, разработанные Бучем (Booch) и Рамбо (Object-Modeling Technique — OMT). OMT был ориентирован на анализ, а Booch — на проектирование программных систем. В октябре 1995 года была выпущена предварительная версия 0.8 унифицированного метода (англ. Unified Method). Осенью 1995 года к компании Rational присоединился Айвар Якобсон, автор метода Object-Oriented Software Engineering — OOSE. OOSE обеспечивал превосходные возможности для спецификации бизнес-процессов и анализа требований при помощи сценариев использования. OOSE был также интегрирован в унифицированный метод.

На этом этапе основная роль в организации процесса разработки UML перешла к консорциуму OMG (Object Management Group). Группа разработчиков в OMG, в которую также входили Буч, Рамбо и Якобсон, выпустила спецификации UML версий 0.9 и 0.91 в июне и октябре 1996 года.

На волне растущего интереса к UML к разработке новых версий языка в рамках консорциума UML Partners присоединились такие компании, как Digital Equipment Corporation, Hewlett-Packard, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle Corporation, Rational Software, Texas Instruments и Unisys. Результатом совместной работы стала спецификация

UML 1.0, вышедшая в январе 1997 года. В ноябре того же года за ней последовала версия 1.1, содержавшая улучшения нотации, а также некоторые расширения семантики.

Последующие релизы UML включали версии 1.3, 1.4 и 1.5, опубликованные, соответственно в июне 1999, сентябре 2001 и марте 2003 года.

Формальная спецификация последней версии UML 2.0 опубликована в августе 2005 года. Семантика языка была значительно уточнена и расширена для поддержки методологии Model Driven Development — MDD (англ.).

UML 1.4.2 принят в качестве международного стандарта ISO/IEC 19501:2005.

Диаграмма классов, Class diagram — статическая структурная диаграмма, описывающая структуру системы, она демонстрирует классы системы, их атрибуты, методы и зависимости между классами.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

- концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
- точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
- точка зрения реализации — диаграмма классов содержит классы, ис пользуемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма разработанной системы представлена на рисунке 4.5.

Класс ActFun (рисунок 4.6) содержит активационные функции используемые в нейронных сетях.

- activation(value:float):static float — активационная функция гиперболи-

ческий тангенс.

Рисунок 4.6 Класс ActFun

Класс Imaging (рисунок 4.7) используется для обработки изображений.

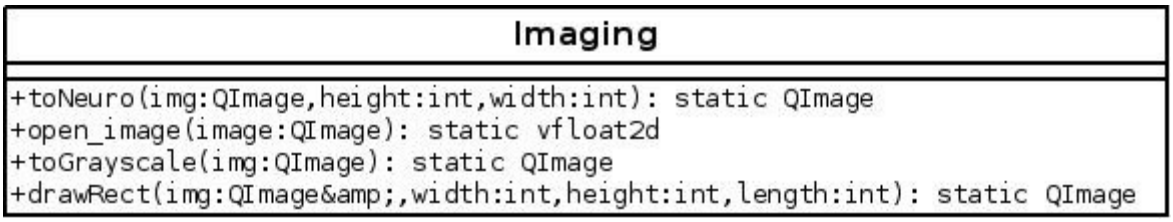

Рисунок 4.7 Класс Imaging

- toNeuro(img:QImage, hight:int, width:int):static Qimage - возвращает изображение уменьшенное в размере и преобразованное к оттенкам серого;
- open_image(image:QImage):static vfloat2d — преобразует изображение в двумерный вектор;
- toGrayscale(img:QImage):static Qimage — преобразует изображение к оттенкам серого;
- drawRect(img:QImage, wight:int, height:int, length:int):static Qimage — рисует прямоугольную рамку заданного размера в определенном месте на изображении.

Класс CNN (рисунок 4.8)реализует сверточную нейронную сеть.

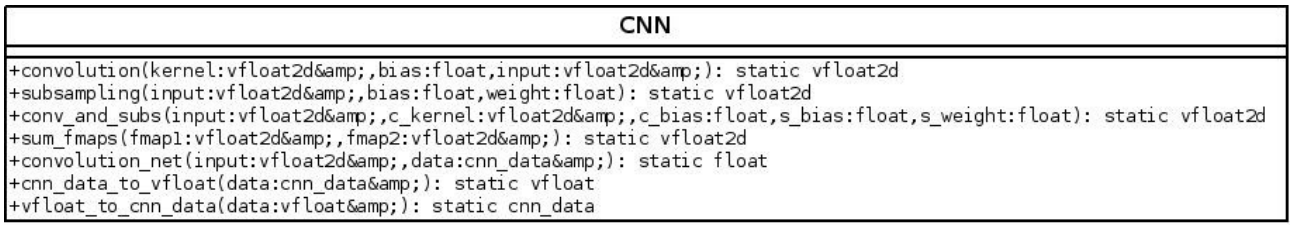

Рисунок 4.8 Класс CNN

- convolution(kernel:vfloat2d, bias:float, input:vfloat2d):static vfloat2d —

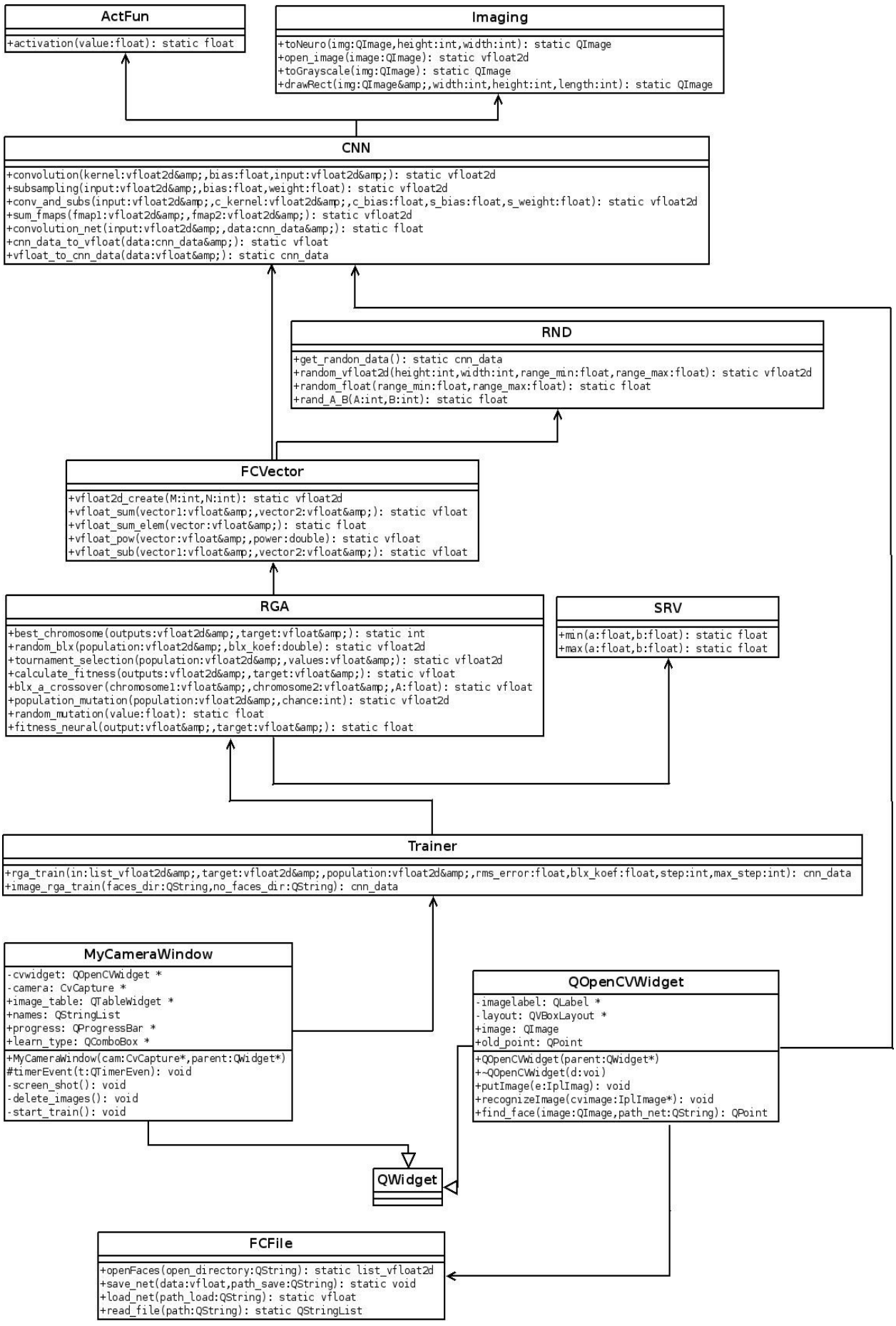

Рисунок 4.5 - UML-диаграмма разработанной системы

сворачивает карту признаков, заданным ядром светки и добавляет нейронное смещение;

- subsampling(input:vfloat2d, bias:float, weight:float):static vfloat2d — функция обеспечивающая локальное усреднение и пространсвенную подвыборку. Карта признаков уменьшается в размере в два раза и пропускается через активационную функцию;
- conv_and_subs(input:vfloat2d, c_kernel:vfloat2d, c_bias:float, s_bias:float, s_weight:float):static vfloat2d — последовательно вызывает функцию свертки и подвыборки;
- sum_fmaps(fmap1:vfloat2d, fmap2:vfloat2d):static vfloat2d — суммирует две карты признаков;
- convolution_net(input:vfloat2d, data:cnn_data):float — описывает процесс функционирования сверточной нейронной сети в режиме прямого распространения;
- cnn_data_to_vfloat(data:cnn_data):static vfloat — возвращает весовые коэффициенты нейронной сети в виде вектора;
- vfloat_to_cnn_data(data:vfloat):static cnn_data — устанавливает веса нейронной сети.

Класс RND (рисунок 4.9) необходим для генерации случайных чисел.

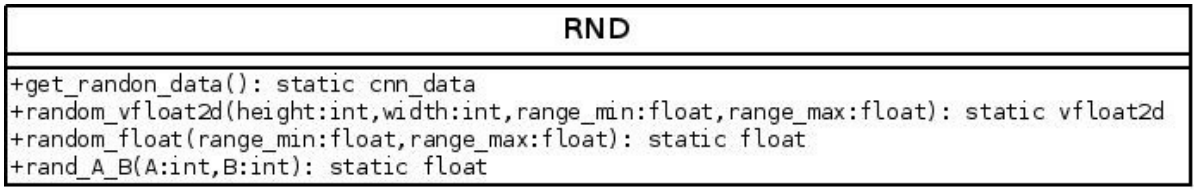

Рисунок 4.9 - Класс RND

- get_random_data():static cnn_data — возвращает весовые коэффициенты сверточной нейронной сети, сгенерированные случайным образом;
- random_vfloat2d(height:int, width:int, range_min:float, range_max:float):

static vfloat2d — генерирует двумерный вектор заданных размеров,

заполненный случайными вещественными числами;

- random_float(range_min:float, range_max:float):static float — генерирует случайное вещественное число из интервала, заданного вещественными числами;
- rand_A_B(A:int, B:int):static float — генерирует случайное число из интервала заданного челыми числами.

Класс FCVector (рисунок 4.10) предоставляет работу с векторами.

- vfloat2d_create(M:int, N:int):static vfloat2d — создает двумерный вектор требуемых размеров;
- vfloat_sum(vector1:vfloat, vector2:vfloat):static vfloat — суммирует два вектора;
- vfloat_sum_elem(vector:vfloat):static float — суммирует все элементы вектора;
- vfloat_pow(vector:vfloat, power:double):static vfloat — возводит все элементы вектора в степень;

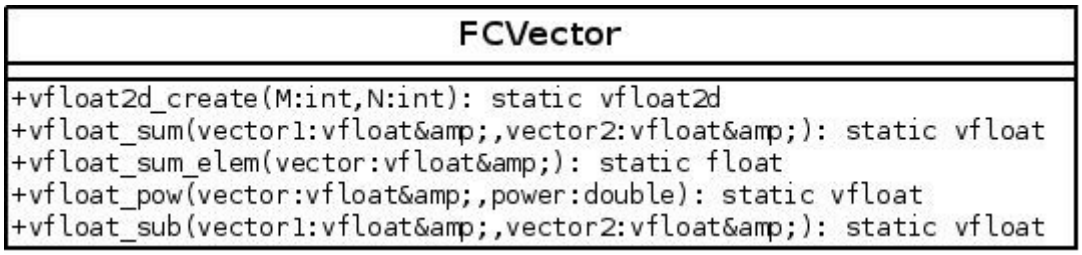

Рисунок 4.10 - Класс FCVector

- vfloat_sub(vector1:vfloat, vector2:vfloat):static vfloat — операция вычитания двух векторов.

Класс RGA (рисунок 4.11) реализует генетические алгоритмы с вещественным кодированием.

- best_chromosome(outputs:vfloat2d, target:vfloat):static int — возвращает индекс хромосомы с лучшей приспособленностью;

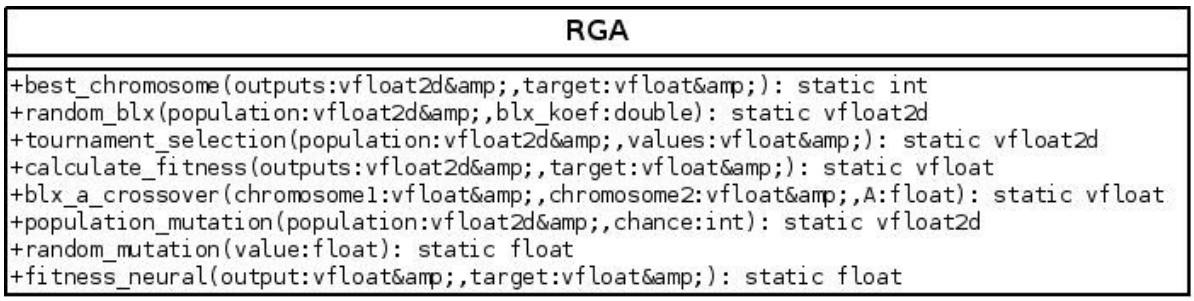

Рисунок 4.11 - Класс RGA

- random_blx(population:vfloat2d, blx_koef:float):static vfloat2d — применяет оператор скрещивания BLX-a ко всем хромосомам в популяции;
- tournament_selection(population:vfloat2d, values:vfloat):static vfloat2d — реализует алгоритм турнирной селекции;
- calculate_fitness(outputs:vfloat2d, target:vfloat):static vfloat — расчитывает приспособленность каждой хромосомы в популяции;
- blx_a_crossover(chromosome1:vfloat, chromosome2:vfloat, A:float):static vfloat — скрещивает две хромосомы BLX-a кроссовером;
- population_mutation(population:vfloat2d, chance:int):static float2d — применяет оператор мутации ко всем хромосомам в популяции;
- random_mutation(value:float):static float — оператор мутации гена в хромосоме;
- fitness_neural(output:vfloat, target:vfloat):static vfloat — рассчитывает приспособленность хромосомы в популяции.

Класс SRV (рисунок 4.12) предоставляет сервисные функции, которые могут понадобиться в различных классах.

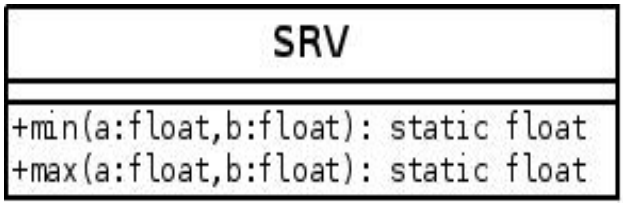

Рисунок 4.12 - Класс SRV

- min(a:float, b:float):static float — возвращает минимальное число среди двух входных аргументов;
- max(a:float, b:float):static float — возвращает максимальное число среди двух входных аргументов.

Класс Trainer (рисунок 4.13) осуществляет обучение нейронной сети.

Рисунок 4.13 - Класс Trainer

- rga_train(in:list_vfloat2d, target:vfloat2d, population:vfloat2d, rms_error:float, blx_koef:float, step:int, max_step:int):cnn_data — подбирает веса нейронной сети управляемым алгоритмом обучения основе генетического поиска и имитации отжига;
- image_rga_train(faces_dir:QString, no_faces_dir:QString):cnn_data — осуществляет открытие и предобработку изображений для последующего обучения нейронной сети.

В классе MyCameraWindow (рисунок 4.14) реализован графический интерфейс пользователя.

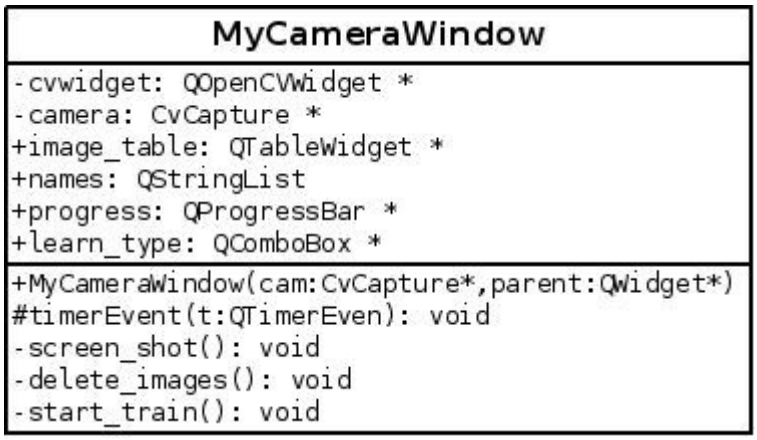

Рисунок 4.14 - Класс MyCameraWindow

- cvWidget:QOpenCVWidget — элемент пользовательского интерфейса отображающий изображение полученное с вебкамеры;

- image_table:QTableWidget — элемент пользовательского интерфейса в котором отображаются снимки сделанные с вебкамеры;
- names:QString — адреса файлов изображений сделанных с вебкамеры.
- progress:QProgressBar — элемент пользовательского интерфейса, отображающий процесс обучения нейронной сети;
- learn_type — элемент пользовательского интерфейса, позволяющий выбрать режим работы программы: обучение или распознавание;
- timer_event(t:QTimerEven):void — таймер по которому срабатывает получение изображение с камеры;
- screen_shot():void — сохраняет выделенный фрагмент изображения;
- delete_images():void — удаляет сохраненные снимки;
- start_train():void — запускает процесс обучения на основе полученных изображений лиц.

Класс QOpenCVWidget (рисунок 4.15)отвечает за отображение данных полученных с вебкамеры.

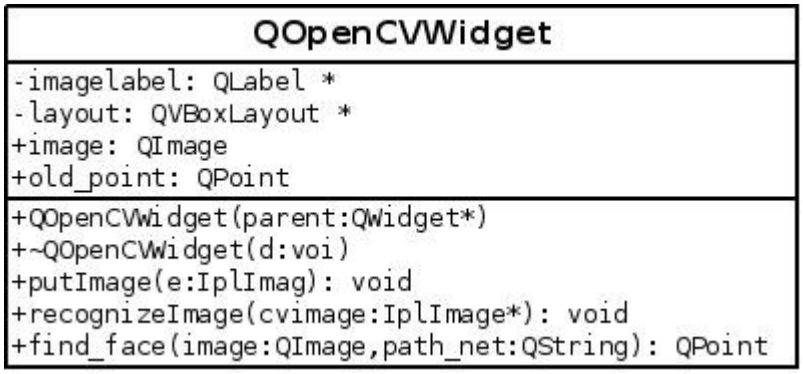

Рисунок 4.15 - Класс QOpenCVWidget

- putImage(e:IplImage):void — размещает исходное изображение полученное с вебкамеры;
- recognizeImage(cvimage:IplImage*):void — отображает изображение с выделенным лицом;

- find_face(image:QImage, path_net:QString):QPoint — ищет изображение лица путем полного перебора всех прямоугольных фрагментов исходного изображения и возвращает координаты найденного лица.

Класс Qwidget (рисунок 4.16) — это базовый класс для всех элементов пользовательского интерфейса.

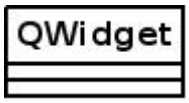

Рисунок 4.16 - Класс QWidget

Класс FCFile (рисунок 4.17) предоставляет возможность работать с файлами на жестком диске.

- openFaces(open_directory:QString):static list_vfloat2d — открывает файлы изображений на жестком диске;
- save_net(data:vfloat, path_save:QString):static void — сохраняет весовые коэффициенты нейронной сети в конфигурационный файл;

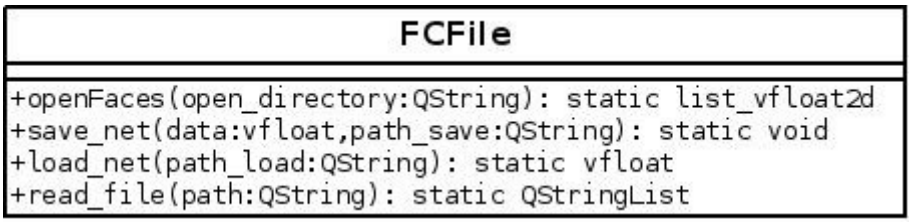

Рисунок 4.17 - Класс FCFile

- load_net(path_load:QString):static vfloat — загружает значения весовых коэффициентов нейронной сети их конфигурационного файла;
- read_file(path:QString) — возвращает полное содержимое текстового файла.

4.3. Разработка пользовательского интерфейса

Эргономика включается в процессы разработки и тестирования программного продукта как часть системы качества. Разработка пользовательского интерфейса (ПИ) ведется параллельно дизайну

программного продукта в целом и в основном предшествует его имплементации. Процесс разработки ПИ разбивается на этапы жизненного цикла:

- Анализ трудовой деятельности пользователя, объединение бизнес функций в роли;
- Построение пользовательской модели данных, привязка объектов к ролям и формирование рабочих мест;
- Формулировка требований к работе пользователя и выбор показателей оценки пользовательского интерфейса;
- Разработка обобщенного сценария взаимодействия пользователя с программным модулем (функциональной модели) и его предварительная оценка пользователями и заказчиком;
- Корректировка и детализация сценария взаимодействия, выбор и дополнение стандарта (руководства) для построения прототипа;
- Разработка макетов и прототипов ПИ и их оценка в деловой игре, выбор окончательного варианта;
- Имплементация ПИ в коде, создание тестовой версии;
- Разработка средств поддержки пользователя (пользовательские словари, подсказки, сообщения, помощь и пр.) и их встраивание в программный код;
- Usability тестирование тестовой версии ПИ по набору раннее определенных показателей;
- Подготовка пользовательской документации и разработка программы обучения.

Эргономические цели и показатели качества программного продукта.

Приложение разрабатывается для обеспечения работы пользователя, т.е.

для того чтобы он с помощью компьютерной программы быстрее и качественнее решал свои производственные задачи.

С точки зрения эргономики, самое важное в программе — создать такой пользовательский интерфейс, который сделает работу эффективной и производительной, а также обеспечит удовлетворенность пользователя от работы с программой.

Эффективность работы означает обеспечение точности, функциональной полноты и завершенности при выполнении производственных заданий на рабочем месте пользователя. Создание ПИ должно быть нацелено на показатели эффективности:

Точность работы определяется тем, в какой степени произведенный пользователем продукт (результат работы), соответствует предъявленным к нему требованиям. Показатель точности включает процент ошибок, которые совершил пользователь: число ошибок набора, варианты ложных путей или ответвлений, число неправильных обращений к данным, запросов и пр.

Функциональная полнота отражает степень использования первичных и обработанных данных, списка необходимых процедур обработки или отчетов, число пропущенных технологических операций или этапов при выполнении поставленной пользователю задачи. Этот показатель может определяться через процент применения отдельных функций в РМ.

Завершенность работы описывает степень исполнения производственной задачи средним пользователем за определенный срок или период, долю (или длину очереди) неудовлетворенных (необработанных) заявок, процент продукции, находящейся на промежуточной стадии готовности, а также число пользователей, которые выполнили задание в фиксированные сроки.

Последовательность действий и набор инструментальных средств пользователя в ПИ должны быть подчинены технологическому процессу выполнения производственного задания.

Не надо бояться сложности системы, надо избегать такого интерфейса, который не соответствует алгоритму решения пользовательских задач.

Необходимо тщательно продумать и осознать сценарий взаимодействия программы с пользователем, приведя его к оптимальной (относительно рассмотренных показателей) системе выполнения задач, и реализовать ПИ в соответствии с этой системой.

Для того, чтобы разобраться в технологии решения задач пользователя, разработчику необходимо выяснить следующие моменты (исследуя деятельность пользователя):

- Какая информация необходима пользователю для решения задачи?
- Какую информацию пользователь может игнорировать (не учитывать)?
- Совместно с пользователем разделить всю информацию на сигнальную, отображаемую, редактируемую, поисковую и результирующую;
- Какие решения пользователю необходимо принимать в процессе работы с программой?
- Может ли пользователь совершать несколько различных действий (решать несколько задач) одновременно?
- Какие типовые операции использует пользователь при решении задачи?
- Что произойдет, если пользователь будет действовать не по предписанному Вами алгоритму, пропуская те или иные шаги или обходя их?

Производительность работы отражает объем затраченных ресурсов при выполнении задачи, как вычислительных, так и психофизиологических.

Дизайн ПИ должен обеспечивать минимизацию усилий пользователя при выполнении работы и приводить к:

- сокращению длительности операций чтения, редактирования и поиска

информации;

- уменьшению времени навигации и выбора команды;
- повышению общей продуктивности пользователя, заключающейся в объеме обработанных данных за определенный период времени;
- увеличению длительности устойчивой работы пользователя и др.

Сокращение непроизводственных затрат и усилий пользователя - важная составляющая качества программного обеспечения.

Для оценки продуктивности используются соответствующие показатели, проверяемые специалистами по эргономике в процессе usability тестирования рабочего прототипа.

Формирование таких показателей происходит в процессе определения требований к ПИ при изучении следующих вопросов:

- Что от пользователя требуется в первую очередь?
- Сколько информации, требующей обработки, поступает пользователю за период времени?
- Каковы требования к точности и скорости ввода информации?
- На какие операции пользователь тратит больше всего времени?
- Чем мы можем облегчить работу пользователя при решении типовых задач?

Удовлетворенность пользователя от работы тесно связана с комфортностью его взаимодействия с приложением, и способствует сохранению профессиональных кадров на предприятии Заказчика за счет привлекательности работы на данном рабочем месте.

Требования к удобству и комфортности интерфейса возрастают с увеличением сложности работ и ответственности пользователя за конечный результат.

Высокая удовлетворенность от работы достигается в случае:

- Прозрачной для пользователя навигации и целевой ориентации в программе. Главное, чтобы было понятно, куда идем, и какую операцию программа после этого шага произведет;
- Ясности и четкости понимания пользователем текстов и значения икон.

В программе должны быть те слова и графические образы, которые пользователь знает или обязан знать по характеру его работы или занимаемой должности.

- Быстроты обучения при работе с программой, для чего необходимо использовать преимущественно стандартные элементы взаимодействия, их традиционное или общепринятое их расположение;
- Наличия вспомогательных средств поддержки пользователя (поисковых, справочных, нормативных), в том числе и для принятия решения в неопределенной ситуации (ввод по умолчанию, обход «зависания» процессов и др.).

Для оценки необходимого уровня удобства интерфейса также используются специальные опросники, формуляры, чек-листы, однако к данной работе лучше привлекать специалистов по эргономике.

Удобный интерфейс помогает пользователю справиться с усталостью и напряжением при работе в условиях высокой ответственности за результат.

Проблемы, возникающие на этапе разработки прототипа GUI и варианты их решения:

а) Учет особенностей устройств ввода/вывода информации, используемых пользователем, например:

- размер экрана монитора;
- разрешение экрана;

- цветовая палитра;
- характеристики звуковой (качество воспроизведения речи) и видеокарты (скорость вывода при анимации);
- вид мыши (с роликом или без);
- тип клавиатуры (“прямая”, “косая”);
- необходимость дополнительного оборудования (штрих-декодера, светового пера сенсорного экрана и др.).

б) Специфика интерактивных элементов, связанная с выбором платформы, стандартных библиотек:

- программная организация ввода/вывода информации;
- изменение и создание новых элементов форм (контролов);
- приобретение нестандартных библиотек у других фирм.

в) Выбор технологии и методов ведения диалога программы с пользователем:

- степень активности пользователя при взаимодействии (автоматический режим или перехват управления программой на себя, визарды, обеспечение доступа ко всем средствам интерфейса независимо от действий пользователя);
- степень учета ситуации (контекстные подсказки, меню дальнейших событий или объектов, запоминание типичных путей диалога)
- соответствие ожиданиям пользователя (предсказание, предобработка, предформатирование);
- устойчивость, терпимость к ошибкам пользователя путем исправления типичных ошибок;
- дублирование вручную отдельных функций системы и дополнительные контрольные процедуры работы отдельных

режимов;

- настройка ПИ на различный уровень подготовки пользователя (образность или метафоричность предметной области в противовес сокращениям и горячим клавишам);
- степень адаптивности ПИ под предпочтения пользователя (изменение способа и порядка отображения, перекомпоновка экрана, выбор отдельных характеристик (стиля) и пр.);
- настройка ПИ на специфику задачи (новый формат данных, изменение набора объектов, дополнение атрибутов объектов).

г) Размещение информации и управляющих элементов в поле экрана, в окне. При композиции экрана необходимо учитывать ограниченные размеры пространства экрана, в связи с чем возникает задача оптимального расположения максимально возможного объема информации путем:

- логической увязкой данных в зависимости от алгоритма работы пользователя, а не ориентацией на структуру и последовательность физических таблиц данных;
- определения уровня “детальности – обобщенности” вывода информации (нахождение компромисса между желанием вывест много записей одновременно и/или сразу увидеть детальную информацию по каждой из них);
- выделения важной информации на экране;
- четкого определения основных и вспомогательных блоков информации;
- определения статических полей на экране, а также полей, где информация периодически изменяется;
- избегания перекрывающихся окон на экране;

- применения принципов гармонии при компоновке экрана (симметрия, баланса масс, соблюдение пропорций, сочетание цветов).

д) Формирование обратной связи между пользователем и приложением:

- показ актуального состояния системы, режима работы системы (автономного, штатного, защищенного и пр.) и режима взаимодействия (например, отображение, редактирование или поиск данных);
- вывод отдельных, важных для рабочей операции данных и показателей;
- отражение действий пользователя (нажатия клавиш, запуск процесса, динамика выполнения процесса, получение ожидаемого и иного результата);
- ясность и информативность сообщений системы.

е) Проектирование панелей меню и инструментов (toolbars) и выбор пунктов в них:

- логическая и смысловая группировка пунктов;
- фиксированная позиция панелей на экране;
- ограничение на ширину списка выборов и шагов (глубины) меню;
- использование привычных названий, широко распространенных икон-пиктограмм, традиционных икон-символов и аккуратное введение сокращений;
- размещение наиболее часто используемых пунктов (обычно в начале списка).

ж) Разработка средств ориентации и навигации:

- легкость определения своего местонахождения и указание

направления следования;

- удобный переход от обобщенного взгляда до конкретных деталей (варьирование степени детализации рассматриваемых объектов);
- быстрый поиск в списке или таблице;
- указание на дополнительно существующую информацию и способ ее получения;
- использование средств листания и прокрутки.

и) Создание форм для ввода данных:

- использования одного или нескольких механизмов ввода в рамках режима (клавиатура, мышь, штрих-декодер, световое перо, др.);
- определение способов ввода данных (таблицы, списки, простая форма, меню и пр.);
- минимизация объема ввода;
- выделение редактируемых обязательных и необязательных, а также нередактируемых полей;
- использование механизмов быстрого ввода (по умолчанию, сокращения, с продолжением и пр.);
- Выделение введенной или отредактированной информации.

Принципы реализации пользовательского интерфейса:

Стилевая гибкость - возможность использовать различные интерфейсы с одним и тем же приложением, на практике реализуется в виде набора “skins”, для web-интерфейсов – с помощью таблицы стилей, в том числе возможность в выборе пользователем собственных установок ПИ (цвет,иконы,подсказки и пр.).

Совместное наращивание функциональности - возможность развивать приложение без разрушения (т.е. оставаясь в рамках) существующего интерфейса.

Масштабируемость - возможность легко настраивать и расширять как интерфейс, так и само приложение при увеличении числа пользователей, рабочих мест, объема и характеристик данных.

Адаптивность к действиям пользователя - приложение должно допускать возможность ввода данных и команд множеством разных способов (клавиатура, мышь, другие устройства) и многовариативность доступа к прикладным функциям (иконы, «горячие клавиши», меню), кроме того программа должна учитывать возможность перехода и возврат от окна к окну, от режима к режиму, и правильно обрабатывать такие ситуации.

Независимость в ресурсах - для создания пользовательского интерфейса должны предоставляться отдельные ресурсы, направленные на хранение и обработку данных, необходимых для поддержки пользователя (пользовательские словари, контекстно-зависимые списки, наборы данных по умолчанию или по последнему запросу, истории запросов и пр.)

Переносимость - при переходе на другую аппаратную (программную) платформу, должен осуществляется автоматически перенос и пользовательского интерфейса, и конечного приложения.