|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

КУРСОВОЙ ПРОЕКТ

|  |
| --- |
| по модулю Проектная деятельность: исследовательский проект |

|  |  |
| --- | --- |
| на тему | Информационное программное обеспечение автоматизированная система |
| обнаружения и классификации посторонних предметов в сыпучих материалах: подсистема | |
| обнаружения | |

|  |
| --- |
| Выполнил студент группы 1ИВТм-01-21оп |
| *группа* |
| направления подготовки (специальности) |
| 09.04.03 Информатика и вычислительная техника |
| *шифр, наименование* |
| **Белов Антон Сергеевич** |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| **Ершов Евгений Валентинович** |
| *фамилия, имя, отчество* |
| **доктор технических наук** |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2020

*Год*

Введение

Актуальность темы. С появлением персональных компьютеров, работа человека упростилась в десятки раз, благодаря автоматизации его рутинных задач. Решение математических задач, задач оптимизации, бухгалтерский учёт и многое другое, все это решает за человека программное обеспечение, установленное на его компьютере.

Нынешние технологии позволяют обрабатывать 100 Тбайт информации за считанные минуты. Это позволяет практически мгновенно анализировать нынешнею ситуацию и принимать соответствующие решения в той или иной области [5]. Но данную информацию необходимо получить из окружающей среды и передать в информационную систему для дальнейшей обработки.

Например, одной из основных задач по созданию и развитию систем автоматизированного управления металлургическим производством является автоматизация процесса обнаружения посторонних предметов в шихтовом материале, перемещающийся в агломерационные печи по конвейерной ленте.

Нынешняя задача на данный момент решается использованием металлоискателей и магнитов, для быстрого извлечения посторонних металлических предметов. Для обнаружения инородных неметаллических материалов прибегают к ручному труду. Недостатком такого решения является, в первую очередь, человеческий фактор, что может привести к порче транспортировочной ленты и/или простою стана. Альтернативным решением является использование машинного обучения.

Машинное обучение (англ. machine learning, ML) — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач [2]. Для построения таких методов используются средства математической статистики, численных методов, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме [11]. Технология машинного обучения – это учитель, который обучает компьютеры выполнению задач на основе данных, а не на базе алгоритма, заложенного программистом.

Такая технология позволит программному обеспечению распознавать посторонние предметы в кадре, для дальнейшего их извлечения из сыпучего материала.

Объектом исследовательской работы является оптико-электронные системы обнаружения и классификации посторонних предметов в сыпучих материалах.

Предметом исследования является математическое, алгоритмическое, программное обеспечение системы обнаружения и классификации посторонних предметов в сыпучих материалах.

В связи с выше сказанным, целью дипломной работы является разработка информационного программного обеспечения автоматизированной системы обнаружения посторонних предметов в сыпучих материалах для уменьшения повреждений транспортировочной ленты, что позволит снизить количество простоев транспортировочных конвейеров агломерационного производства и устранить причины попадания посторонних предметов в сыпучие материалы за счёт их классификации.

Для поставленной цели определен ряд задач, которые необходимо выполнить для её достижения:

* проанализировать предметную область;
* теоретически обосновать решаемую задачу. Описать методы и модели. Разработать алгоритмы;
* спроектировать программное и информационное обеспечение автоматизированной системы обнаружения посторонних предметов в сыпучих материалах. Разработать логические и физические модели. Разработать модули программного обеспечения;
* провести экспериментальную проверку и получить результаты предложенных технических решений.

Методы исследования. Для решения поставленных задач в рамках магистерской диссертации были использованы методы машинного обучения.

Научная новизна работы заключается в разработке нового алгоритма программного обеспечения, работающего с использованием методов машинного обучения, позволяющее эффективно проводить распознавание посторонних предметов в любых сыпучих материалах.

Практическая значимость результата работы:

* уменьшение повреждений транспортировочной ленты;
* снижение количество простоев транспортировочных конвейеров агломерационного производства;
* устранение причин попадания посторонних предметов в сыпучие материалы.

1. Характеристика проблемы обнаружения посторонних объектов в сыпучих материалах на основе методов машинного обучения в системах автоматического контроля
   1. Анализ предметной области

ПАО «Северсталь» — российская вертикально интегрированная сталелитейная и горнодобывающая компания, территориально располагающаяся по западной части России и зарубежным странам. Но центром является Череповецкая площадка – Череповецкий металлургический комбинат (ЧерМК), один из самых рентабельных металлургических предприятий в мире. ЧерМК выпускает самый широкий среди российских металлургических предприятий сортамент продукции. Продуктовая линейка предприятия постоянно расширяется, при этом акцент делается на разработке высокотехнологичных продуктов с высокой добавленной стоимостью с учетом требований клиентов [17].

Предприятие имеет полный металлургический цикл, включая коксохимическое, агломерационное, доменное, сталеплавильное и прокатное производства. Это гарантирует сквозной контроль производства и качества продукции.

С 2018 года компания ПАО Северсталь объявила о информационной трансформации, которая подразумевает внедрение IT-технологий во все участки производства конечного продукта. Агломерационное производство не стало исключением.

Агломерация (от лат. agglomero — присоединяю, накопляю) — метод термического окускования пылеватых мелких руд, концентратов и металлосодержащих отходов путём их спекания. Наиболее широко агломерация применяется для подготовки железорудного сырья для металлургического производства чугуна.

Агломерация железорудного концентрата (иногда в смеси с рудой, отходами металлургического производства) является заключительной операцией в комплексе мероприятий по подготовке железных руд к доменной плавке. Главная цель этой операции состоит в том, чтобы превратить мелкий рудный концентрат в более крупные куски — агломерат, использование которого в доменной плавке обеспечивает формирование слоя шихты хорошей газопроницаемости, что является непременным условием высокопроизводительной работы доменной печи [16].

Доменная плавка высокой интенсивности возможна при большом количестве сгорающего в горне доменной печи кокса, что, с одной стороны, ведет к выделению большого количества тепла, а с другой — к образованию в нижней части печи свободного пространства (благодаря газификации твердого кокса), куда опускается столб доменной шихты.

Шихта (нем. Schicht) — смесь исходных материалов, а в некоторых случаях (например, при выплавке чугуна в доменной печи) и топлива в определённой пропорции, подлежащая переработке в металлургических, химических и других агрегатах.

Шихтами обычно называют исходные смеси, использующиеся в пирометаллургических или иных высокотемпературных процессах (производство стекла, цемента, некоторых керамических материалов и изделий, коксование углей). Шихта рассчитана на получение конечных продуктов с заданными физическими и химическими свойствами. Шихта загружается в агрегат либо в виде однородной смеси (порошковой, кусковой, брикетированной), приготовленной вне агрегата, либо порциями, состоящими из отдельных компонентов шихты [17].

Типичная шихта, идущая на производство железорудного агломерата, состоит из следующих компонентов:

* мелкий железорудный материал, как правило, концентрат;
* измельченное топливо — кокс (фракция 0—3 мм), содержание в шихте 4—6 %;
* измельченный известняк (фракция 0—3 мм), содержание до 8—10 %;
* возврат — некондиционный агломерат от предыдущего спекания (фракция 0—8 мм), содержание 25—30 %;
* железосодержащие добавки — колошниковая пыль из доменных печей, окалина прокатных цехов, пиритные огарки сернокислотного производства и др. (фракция 0—3 мм), содержание до 5 %.

Отдозированные в заданном соотношении компоненты перемешивают, увлажняют (для улучшения окомкования) и после окомкования без уплотнения загружают на колосниковую решетку слоем 300—400 мм. Затем включают нагнетатель — вентилятор, работающий на отсос. Под колосниковой решеткой создается разрежение, благодаря которому в слой вначале засасывается поток горячих горновых газов, обеспечивающих «зажигание» шихты, то есть происходит нагрев поверхностного слоя примерно до 1200 °С (в течение 1,5 - 2,0 мин). Поступающий затем в слой в остальное время процесса атмосферный воздух обеспечивает интенсивное горение частиц кокса шихты. В зоне максимальных температур (1400 - 1450°С) происходит частичное плавление рудных зерен, их слипание, а затем в ходе последующей кристаллизации образуется пористая структура — агломерационный спёк [17].

Шихтовые материалы перед загрузкой в агрегат, в компании, хранятся на шихтовом дворе, куда его привозят КАМАЗы. Далее, машинист бульдозера перенаправляет шихту в бункер, по которому она попадает на транспортировочною ленту (рис.1), для перехода в агрегат.



Рис. 1. Транспортировочная лента

Проблема заключается в том, что КАМАЗы, привозящие шихту, в кузове, работают в разных частях ЧерМК. В связи с этим в кузовах машины могу оставаться посторонние предметы, например, проволока, профиль, обломки листов металла и многое другое, которые попадают в машину с других переделов, так как компания не имеет своего штата сотрудников и техники, а работают с подрядчиком, которые в течении дня работают по всей территории завода. Большинство таких предметов при попадании далее на транспортировочную ленту вместе с шихтой могут повредить ее. Самым тяжелым повреждением является ее обрыв.

Ремонт транспортировочных лент можно осуществлять разными способами (в зависимости от условий и возможности) [17]:

* ремонт методом горячей вулканизациии резинотросывых и резинотканевых лент;
* ремонт методом холодной вулканизациии резинотросовых конвейерных лент;
* ремонт механическими скобами (рис.2.).

Рис.2. Примеры механических скоб

Самый дешёвый и быстры способ ремонта – это использование механических скоб, которыми соединяют две части ленты. Но у него есть важный недостаток – время, затраченное мастерами на ремонт, а также данный способ восстановления работоспособности ленты подразумевает лишь кратковременный аварийный ремонт, так как требует либо замены ленты, либо ремонт методом горячей/холодной вулканизациии.

* 1. Сравнительный анализ методов и средств решения задачи локализации объектов в системах автоматического контроля
     1. Анализ методов машинного обучения

Существует множество методов классификации, которые используют различный математический аппарат и различные подходы при реализации. Однако эффективность этих методов зависит от конкретной решаемой задачи.

Можно выделить следующие типы методов классификации: вероятностные, метрические, логические, линейные, логическая регрессия [2]. Обобщенно опишем некоторые из них, указывая преимущества и недостатки каждого из них.

Метод Байеса (Naive Bayes, NB) относится к вероятностным методам классификации. Преимущества метода состоит в следующем: высокая скорость работы, поддержка инкрементного обучения, простая реализация алгоритма в виде программы, легкая интерпретируемость результатов работы алгоритма. Несмотря на приведенные достоинства, метод Байеса имеет так же и минусы в своей реализации. Относительно низкое качество классификации и неспособность учитывать зависимость результата классификации от сочетания признаков являются главными недостатками этого метода [13].

Метод k ближайших соседей (k Nearest Neighbors, KNN) относится к метрическим методам и считается простейшим классификатором. Объект присваивается тому классу, который является наиболее распространенным среди соседей данного элемента. Достоинства данного метода: простая реализация, проработанная теоретическая база, адаптация под нужную задачу выбором метрики или ядра, интерпретируемость. К недостаткам относятся: недостаточная производительность в реальных задачах, так как число соседей, используемых для классификации, будет достаточно большим; трудность в наборе подходящих весов и определением, какие признаки необходимы для классификации; зависимость от выбранной метрики расстояния между примерами [13].

Метод опорных векторов (Support Vector Machine, SVM) является линейным методом классификации, в настоящее время призван одним из лучших. Потенциальные недостатки метода опорных векторов заключается в следующем: невозможность калибровки вероятности попадания в определенный класс, подходит только для решения задач с 2 классами, параметры модели сложно интерпретировать [13].

Метод деревьев решений (Decision Trees, DT) относится к логическим методам классификации. Деревом решений называют ациклический граф, по которому производится классификация документов, описанных набором признаков. Каждый узел дерева содержит условие ветвления по одному из признаков. У каждого узла столько ветвлений, сколько значений имеет выбранный признак. Главным преимуществом метода является высокая производительность обучения и прогнозирования, такие деревья решений можно легко визуализировать и интерпретировать [13].

Нейронные сети активно используются в связи с появлением больших объемов данных и больших вычислительных возможностей. Сети с прямой связью являются универсальным средством аппроксимации функций, что позволяет их использовать при решении задач классификации [4]. Их эффективность достаточно высока, потому что они генерируют фактически большое число регрессионных моделей (которые используются в решении задач классификации статистическими методами). Однако, любой метод, основанный на нейронных сетях, никогда не даст классификатор нужного качества, если имеющий набор примеров не будет достаточно полным для той задачи, с которой придется работать в системе.

Начиная с 2016 года нейронные сети шагнули вперед в своем развитии по следующим направлениям [4]:

* системы распознавания и классификации объектов на изображениях;
* голосовые интерфейсы взаимодействия для интернета вещей;
* системы мониторинга качества обслуживания в колл-центрах;
* системы выявления неполадок (в том числе, предсказывающие время технического обслуживания), аномалий, кибер-физических угроз;
* системы интеллектуальной безопасности и мониторинга;
* замена ботами части функций операторов колл-центров;
* системы видеоаналитики;
* самообучающиеся системы, оптимизирующие управление материальными потоками или расположение объектов (на складах, транспорте);
* интеллектуальные, самообучающиеся системы управления производственными процессами и устройствами (в том числе, робототехнические);
* появление систем универсального перевода «на лету» для конференций и персонального использования;
* появление ботов-консультантов технической поддержки или персональных ассистентов, по функциям близким к человеку.

Именно нейронные сети позволят решить задачу распознавание и классификация объектов на изображении, для дальнейшего подсчёта вагонов, входящих в состав. Анализ типов нейронных сетей приведен в следующем пункте.

* + 1. Анализ типов нейронных сетей

Нейронные сети сейчас постоянно получаю архитектурное развитие, создаются новые, совершенствуются старые. В основном выделяют 13 типов сетей, рассмотренные далее [1].

В каждой модели нейронной сети нейроны обозначены в соответствии с рис.1.



Рис.1. Обозначение нейронов в моделях нейронных сетей



Рис.2. Нейронные сети прямого распространения

Нейронные сети прямого распространения (feed forward neural networks, FF или FFNN) и перцептроны (perceptrons, P) очень прямолинейны, они передают информацию от входа к выходу (рис.2, 3). Нейронные сети часто описываются в виде слоёного торта, где каждый слой состоит из входных, скрытых или выходных клеток. Клетки одного слоя не связаны между собой, а соседние слои обычно полностью связаны. Самая простая нейронная сеть имеет две входных клетки и одну выходную, и может использоваться в качестве модели логических вентилей. FFNN обычно обучается по методу обратного распространения ошибки, в котором сеть получает множества входных и выходных данных. Этот процесс называется обучением с учителем, и он отличается от обучения без учителя тем, что во втором случае множество выходных данных сеть составляет самостоятельно. Вышеупомянутая ошибка является разницей между вводом и выводом. Если у сети есть достаточное количество скрытых нейронов, она теоретически способна смоделировать взаимодействие между входным и выходными данными. Практически такие сети используются редко, но их часто комбинируют с другими типами для получения новых.

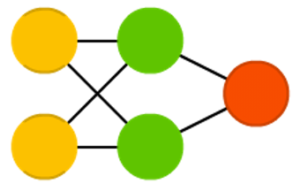


Рис.3. Нейронные сети радиально-базисных функций

Сети радиально-базисных функций (radial basis function, RBF) — это FFNN, которая использует радиальные базисные функции как функции активации (рис.3.). В остальном – это нейронная сеть прямого распространения.

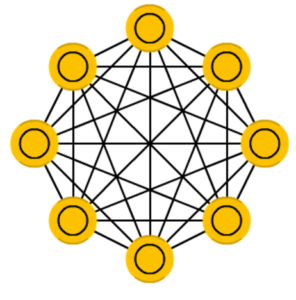


Рис.4. Нейронная сеть Хопфилда

Нейронная сеть Хопфилда (Hopfield network, HN) — это полносвязная нейронная сеть с симметричной матрицей связей (рис.4). Во время получения входных данных каждый узел является входом, в процессе обучения он становится скрытым, а затем становится выходом. Сеть обучается так: значения нейронов устанавливаются в соответствии с желаемым шаблоном, после чего вычисляются веса, которые в дальнейшем не меняются. После того, как сеть обучилась на одном или нескольких шаблонах, она всегда будет сводиться к одному из них (но не всегда — к желаемому). Она стабилизируется в зависимости от общей «энергии» и «температуры» сети. У каждого нейрона есть свой порог активации, зависящий от температуры, при прохождении которого нейрон принимает одно из двух значений (обычно -1 или 1, иногда 0 или 1). Такая сеть часто называется сетью с ассоциативной памятью; как человек, видя половину таблицы, может представить вторую половину таблицы, так и эта сеть, получая таблицу, наполовину зашумленную, восстанавливает её до полной.

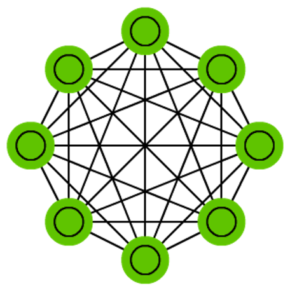


Рис.5. Цепи Маркова

Цепи Маркова (Markov chains, MC или discrete time Markov Chains, DTMC) — это предшественники машин Больцмана (BM) и сетей Хопфилда (HN) (рис.5). Их смысл можно объяснить так: каковы мои шансы попасть в один из следующих узлов, если я нахожусь в данном? Каждое следующее состояние зависит только от предыдущего. Хотя на самом деле цепи Маркова не являются НС, они весьма похожи. Также цепи Маркова не обязательно полносвязны.

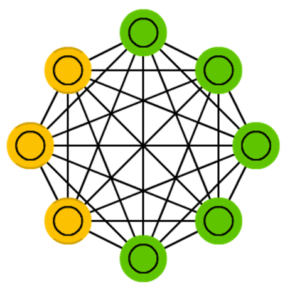


Рис.6. Машина Больцмана

Машина Больцмана (Boltzmann machine, BM) очень похожа на сеть Хопфилда, но в ней некоторые нейроны помечены как входные, а некоторые — как скрытые (рис.6). Входные нейроны в дальнейшем становятся выходными. Машина Больцмана — это стохастическая сеть. Обучение проходит по методу обратного распространения ошибки или по алгоритму сравнительной расходимости. В целом процесс обучения очень похож на таковой у сети Хопфилда.

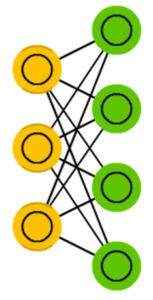


Рис.7. Ограниченная машина Больцмана

Ограниченная машина Больцмана (restricted Boltzmann machine, RBM) удивительно похожа на машину Больцмана и, следовательно, на сеть Хопфилда (рис.7). Единственной разницей является её ограниченность. В ней нейроны одного типа не связаны между собой. Ограниченную машину Больцмана можно обучать как FFNN, но с одним нюансом: вместо прямой передачи данных и обратного распространения ошибки нужно передавать данные сперва в прямом направлении, затем в обратном. После этого проходит обучение по методу прямого и обратного распространения ошибки.

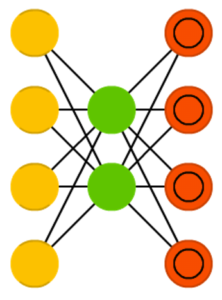


Рис.8. Автокодировщик

Автокодировщик (autoencoder, AE) чем-то похож на FFNN, так как это скорее другой способ использования FFNN, нежели фундаментально другая архитектура (рис.8). Основной идеей является автоматическое кодирование (в смысле сжатия, не шифрования) информации. Сама сеть по форме напоминает песочные часы, в ней скрытые слои меньше входного и выходного, причём она симметрична. Сеть можно обучить методом обратного распространения ошибки, подавая входные данные и задавая ошибку равной разнице между входом и выходом.

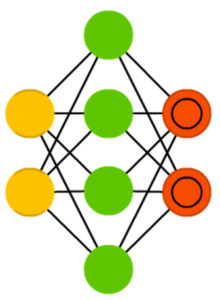


Рис.9. Разреженный автокодировщик

Разреженный автокодировщик (sparse autoencoder, SAE) — в каком-то смысле противоположность обычного (рис.9). Вместо того, чтобы обучать сеть отображать информацию в меньшем «объёме» узлов, мы увеличиваем их количество. Вместо того, чтобы сужаться к центру, сеть там раздувается. Сети такого типа полезны для работы с большим количеством мелких свойств набора данных. Если обучать сеть как обычный автокодировщик, ничего полезного не выйдет. Поэтому кроме входных данных подаётся ещё и специальный фильтр разреженности, который пропускает только определённые ошибки.

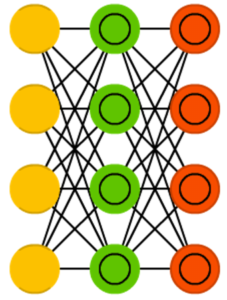


Рис.10. Вариационные автокодировщики

Вариационные автокодировщики (variational autoencoder, VAE) обладают схожей с AE архитектурой, но обучают их иному: приближению вероятностного распределения входных образцов (рис.10). В этом они берут начало от машин Больцмана. Тем не менее, они опираются на байесовскую математику, когда речь идёт о вероятностных выводах и независимости, которые интуитивно понятны, но сложны в реализации. Если обобщить, то можно сказать что эта сеть принимает в расчёт влияния нейронов. Если что-то одно происходит в одном месте, а что-то другое — в другом, то эти события не обязательно связаны, и это должно учитываться.

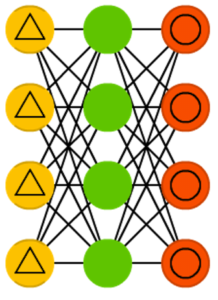


Рис.11. Шумоподавляющие автокодировщики

Шумоподавляющие автокодировщики (denoising autoencoder, DAE) — это AE, в которые входные данные подаются в зашумленном состоянии (рис.11). Ошибку мы вычисляем так же, и выходные данные сравниваются с зашумленными. Благодаря этому сеть учится обращать внимание на более широкие свойства, поскольку маленькие могут изменяться вместе с шумом.

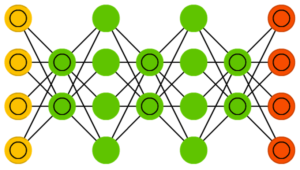


Рис.12. Сеть типа «deep belief»

Сеть типа «deep belief» (deep belief networks, DBN) — это название, которое получил тип архитектуры, в которой сеть состоит из нескольких соединённых RBM или VAE (рис.12). Такие сети обучаются поблочно, причём каждому блоку требуется лишь уметь закодировать предыдущий. Такая техника называется «жадным обучением», которая заключается в выборе локальных оптимальных решений, не гарантирующих оптимальный конечный результат. Также сеть можно обучить (методом обратного распространения ошибки) отображать данные в виде вероятностной модели. Если использовать обучение без учителя, стабилизированную модель можно использовать для генерации новых данных.

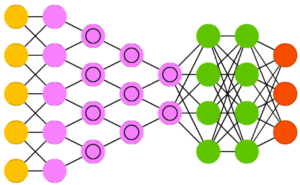


Рис.13. Свёрточные нейронные сети

Свёрточные нейронные сети (convolutional neural networks, CNN) и глубинные свёрточные нейронные сети (deep convolutional neural networks, DCNN) сильно отличаются от других видов сетей (рис.13). Обычно они используются для обработки изображений, реже для аудио. Типичным способом применения CNN является классификация изображений: если на изображении есть кошка, сеть выдаст «кошка», если есть собака — «собака». Такие сети обычно используют «сканер», не парсящий все данные за один раз. Например, если у вас есть изображение 200×200, вы не будете сразу обрабатывать все 40 тысяч пикселей. Вместо это сеть считает квадрат размера 20 x 20 (обычно из левого верхнего угла), затем сдвинется на 1 пиксель и считает новый квадрат, и т.д. Эти входные данные затем передаются через свёрточные слои, в которых не все узлы соединены между собой. Эти слои имеют свойство сжиматься с глубиной, причём часто используются степени двойки: 32, 16, 8, 4, 2, 1. На практике к концу CNN прикрепляют FFNN для дальнейшей обработки данных. Такие сети называются глубинными (DCNN).

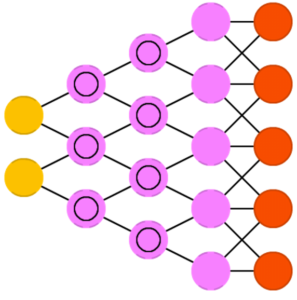


Рис.14. Развёртывающие нейронные сети

Развёртывающие нейронные сети (deconvolutional networks, DN), также называемые обратными графическими сетями, являются обратным к свёрточным нейронным сетям (рис.14). Представьте, что вы передаёте сети слово «кошка», а она генерирует картинки с кошками, похожие на реальные изображения котов. DNN тоже можно объединять с FFNN. Стоит заметить, что в большинстве случаев сети передаётся не строка, а какой бинарный вектор: например, <0, 1> — это кошка, <1, 0> — собака, а <1, 1> — и кошка, и собака.

Из всего многообразия видов нейронных сетей, лучшим для решения поставленной задачи являются сверточные нейронные сети, которые рассмотрены в следующем пункте.

* + 1. Анализ архитектур многослойных сверточных нейронных сетей

LeNet – первая сверточная нейронная сеть, предложенная Яном Лекуном, состоит из трёх типов слоёв: свёрточные (convolutional) слои, субдискретизирующие (subsampling, подвыборка) слои и слои "обычной" нейронной сети – перцептрона. Первые два типа слоёв (свёрточные, субдискретизирующие), чередуясь между собой, формируют входной вектор признаков для многослойного перцептрона. Сеть можно обучать с помощью градиентных методов. Данная архитектура без значительных изменений до сих пор в большинстве случаев используется при решении прикладных задач медицинской диагностики на изображениях и видеоданных [19].

AlexNet или DarkNet (Алекс Крижевский) – данная сеть очень похожа на сеть LeNet, однако отличалась от неё более массивной и сложной архитектурой. Также она имела всего лишь один сверточный слой и несколько слоев субдискретизации по принципу выбора максимального значения. Сверточная нейронная сеть AlexNet предназначена для распознавания объектов любой сложности на больших изображениях размером 224 х 224 [19].

ZF Net (Метью Цейлер, Робер Фергюс) - Данная сеть представляет собой модификацию сети AlexNet, основная особенность которой заключается в более удачном наборе параметров сети: увеличение размеров внутренних сверточных слоев сети, а также уменьшение размеров смещения и размеров фильтров в первом сверточном слое [19].

GoogLeNet (Google) - данная сверточная сеть является очень глубокой – до 22 слоев. Но, несмотря на это, имеет в 10 раз меньше параметров, чем сеть AlexNet, что положительно сказывается на производительности и расходовании памяти. Также в ней используются малые размеры фильтров, а слой субдискретизации реализован по принципу выбора среднего значения [19].

VGGNet (Карен Симонян, Эндрю Зиссерман) - Данная СНС, также как и GoogLeNet, является очень глубокой (до 16 слоев) и состоит из большого количества чередующихся слоев свертки и субдискретизации, имеющих малые размеры (3 х 3 – размер фильтров сверточного слоя, 2 х 2 – размер окон слоя субдискретизации). Отрицательной стороной данной сети является то, что она хранит до 140 миллионов параметров, что делает её громоздкой и низко-производительной [19].

ResNet (Microsoft Research Asia) - архитектура данной сети предполагает большое количество сверточных слоев, содержащих большое количество (до 512) фильтров малого размера (3 х 3). Глубина сети может достигать 152 слоев. На примере данной сети было установлено, что СНС может использовать только сверточные слои и качество распознавания значительно увеличивается при увеличении глубины сети. Данная сеть является одной из наиболее эффективных сверточных нейронных сетей на сегодняшний день [19].

Для решения задачи распознавания и классификации подойдет архитектура AlexNet, которая позволит быстро производить анализ видеопотока на не мощном оборудовании, например, видеокарте NVIDIA GTX 960, примерно со скоростью 20-25 кадров в секунду, с минимальной вероятностью появления ложных распознаваний [1].

Следующим пунктом рассмотрены нынешние аппаратно-программные комплексы (АПК) на рынке, работающие на основе нейронных сетей, со схожей архитектурой.

* + 1. Анализ существующих аппаратно-программных средств решения

Нейросетевые алгоритмы находят широкое применение в вычислительной практике. В частности, особое внимание уделяется классификации и кластеризации изображений, распознаванию речи и изображений, прогнозу финансовых показателей, искусственному синтезу речи, аппроксимации функционалов, совершенствованию методов электроимпедансной и магнитоиндукционной томографии, анализу данных (data mining) [9].

Известным ограничением развития нейросетевых алгоритмов следует признать высокие вычислительные затраты на реализацию таких методов [2]. К традиционным способам решения данной проблемы относят организацию параллельных и распределенных вычислений на специализированном аппаратном обеспечении, таком как нейронные чипы, систолические нейропроцессоры, ПЛИС, распределенные кластерные системы [3], GRID-технологии. Появившийся недавно аппаратно-программный комплекс CUDA (Compute Unified Device Architecture) позволяет использовать процессоры видеокарт (GPU) как ускорители научных и инженерных расчетов и проводить вычисления, по эффективности сравнимые с современными кластерными системами [4][5]. Принципиальное отличие архитектур состоит в следующем: исполнение команд на кластере происходит в стиле MIMD (много потоков команд, много потоков данных – когда набор процессоров независимо выполняет различные наборы команд, обрабатывающих различные наборы данных), а в среде CUDA характеризуется стилем SIMD (один поток команд, много потоков данных – когда несколько процессоров исполняют одну и ту же команду над разными данными, обычно элементами массива). Особенностью оборудования, поддерживающего технологию CUDA, является возможность обеспечивать на порядок большую (по сравнению с кластерами) пропускную способность при работе с памятью. В частности, при вычислении кулоновской ионизации сверхбольших структур, таких как вирусы, может потребоваться несколько дней работы кластера средних размеров, в то время как решение той же задачи на персональном компьютере, оснащенном видеокартой с поддержкой CUDA, можно провести менее чем за час [6].

В настоящее время аппаратная часть CUDA присутствует во многих (в том числе и бюджетных) видеокартах NVIDIA. Таким образом, для полноценной работы с нейронными сетями следует установить ПО NVIDIA CUDA, которая обеспечивает возможность выполнения на GPU любых вычислений, либо библиотеку cuDNN, которая специально разработана для обучения глубоких нейронных сетей. Она содержит оптимизированные для GPU реализации свёрточных и рекуррентных сетей, различных функций активации (полулинейная, сигмоидальная, гиперболический тангенс, softmax), алгоритма обратного распространения ошибки и т.п. cuDNN позволяет обучать нейронные сети на GPU в несколько раз быстрее, чем просто CUDA. Библиотека cuDNN содержит оптимизированные для GPU функции обучения нейронных сетей. Она позволяет обучать нейронную сеть в несколько раз быстрее, чем с использованием CUDA, и в несколько десятков раз быстрее, чем на CPU [4][6].

OpenCV — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD [5].

Keras — открытая нейросетевая библиотека, написанная на языке Python. Она представляет собой надстройку над фреймворками Deeplearning, TensorFlow и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System), а ее основным автором и поддерживающим является Франсуа Шолле, инженер Google. Эта библиотека содержит многочисленные реализации широко применяемых строительных блоков нейронных сетей, таких как слои, целевые и передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом [5][6].

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия [6].

Автомаршал – программное обеспечение для автоматического распознавания номеров автомобилей в транспортном потоке и в пунктах контроля (рис.15) [20]. Применяется для автоматизации работы парковок, КПП и проходных предприятий, логистических и складских комплексов, учета транспорта на автомагистралях, весовых, автомойках и заправочных станциях, а также для управления доступом автомобилей на территорию многоквартирных жилых домов и загородных поселков.

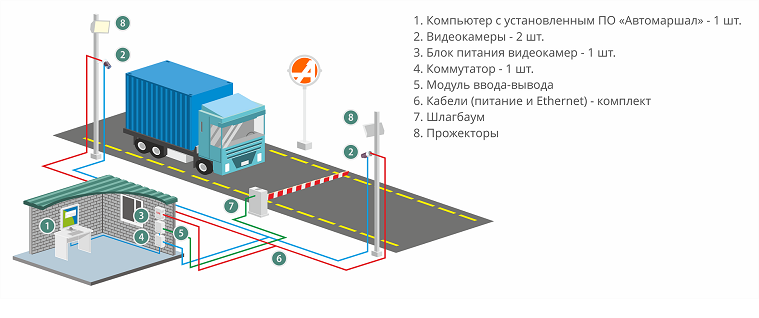


Рис.15. Модель АПК «Автомаршал»

Traffix - программное обеспечение предназначено для подсчета интенсивности и определения состава трафика по видеоизображению (рис.16) [20]. Видео поступает непосредственно с обзорных и специальных камер онлайн, либо может быть загружено из видеофайла. В результате анализа программа выдает статистику по количеству транспортных средств и их категориям.

Адаптивная смарт-камера VIRIS с высоким уровнем распознавания номеров автомобилей [20]. Предлагаемая технологическая новинка объединяет в себе возможности видеокамеры и ПК, а специальный интеллектуальный модуль обеспечивает адаптацию устройства к различным изменениям внешней среды.

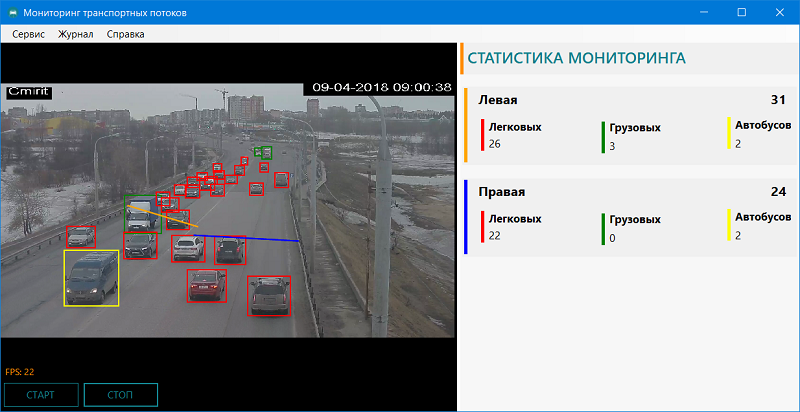


Рис.16. Программа Traffix

АРСИС (ARSCIS) – это семейство решений по идентификации вагонов, управлению отгрузкой продукции ж/д транспортом, коммерческому осмотру вагонов, контролю передвижения и местонахождения вагонов, построенное на платформе системы распознавания номеров вагонов и решающее задачи служб метрологии, логистики и безопасности (рис.16) [20].

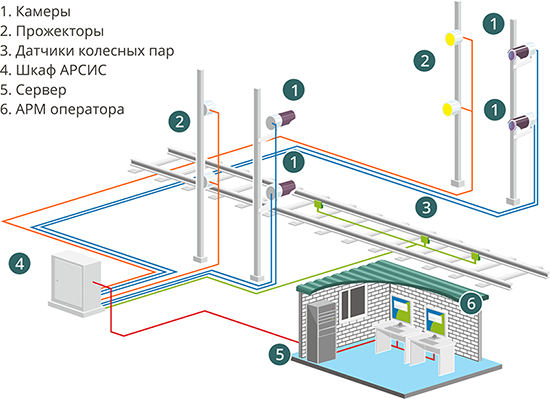


Рис.17. АПК «АРСИС»

Road Traffic Analysis – программное обеспечение, предназначенное для анализа дорожного трафика, по видеоизображени: интенсивность движения, скорость автомобилей, плотность трафика и т.п. (рис.17) [20].

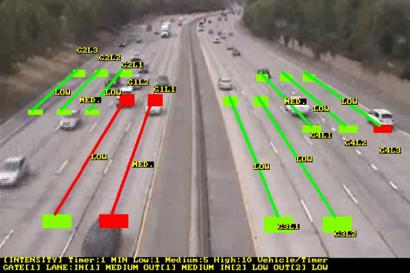


Рис.18. Программа Road Traffic Analysis

Все эти и другие программные решения выполняют конкретную задачу, например, распознавание номеров на цистернах, или классификация номеров. Нет на рынке готово решения, которое бы позволило вести подсчет вагонов по видеоизображения.

Для разработки программного обеспечения, решающего поставленную задачу, необходимо проанализировать объекты, которые будут находиться в кадре, получаемое с обзорных видеокамер. Описание представлено в следующем пункте.

* 1. Описание объектов на изображении для решения задачи

Подумать и пописать

1. Математическое обеспечение локализации объектов в системах автоматизированного контроля
   1. Описание архитектуры сверточной нейронной сети

DarkNet хранит обученные коэффициенты (веса) в формате, который может быть распознан с помощью различных методов на различных платформах. DarkNet написан на языке C и не имеет другого программного интерфейса [9][20].

DarkNet состоит из разных видов слоев: сверточные (convolutional) слои, субдискретизирующие (subsampling, подвыборка) слои, и слои «обычной» нейронной сети – персептрона, в соответствии с рис.23.

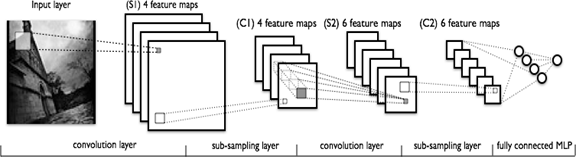
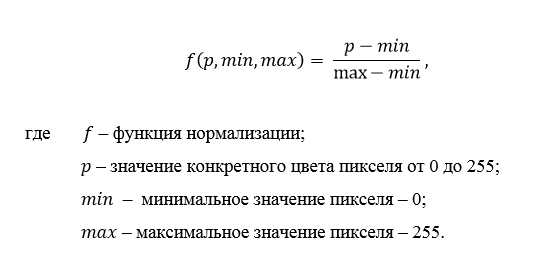


Рис.23. Топология нейронной сети DarkNet

Первые два типа слоев (convolutional, subsampling), чередуясь между собой, формируют входной вектор признаков для многослойного персептрона.

Входные данные представляют из себя цветные изображения типа JPEG, размера 480х480 пикселей. Каждое изображение разбивается на 3 канала: красный, синий, зеленый. Таким образом получается 3 изображения размера 480х480 пикселей. Входной слой учитывает двумерную топологию изображений и состоит из нескольких карт (матриц), карта может быть одна, в том случае, если изображение представлено в оттенках серого, иначе их 3, где каждая карта соответствует изображению с конкретным каналом (красным, синим и зеленым).

Входные данные каждого конкретного значения пикселя нормализуются в диапазон от 0 до 1, по формуле:



Сверточный слой представляет из себя набор карт (другое название – карты признаков, в обиходе это обычные матрицы) (рис.24), у каждой карты есть синоптическое ядро (в разных источниках его называют по-разному: сканирующее ядро или фильтр).

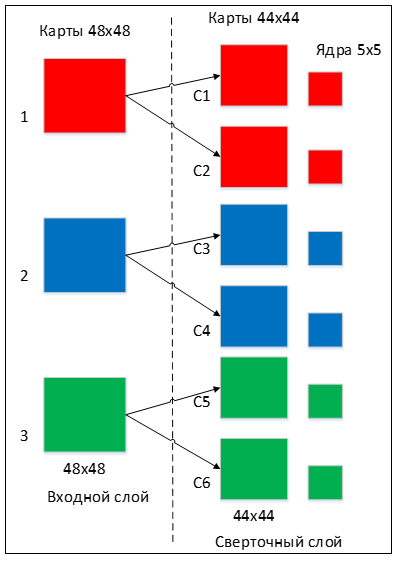
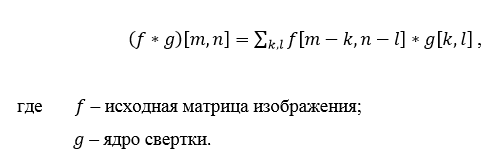


Рис.24. Представление сверточного слоя

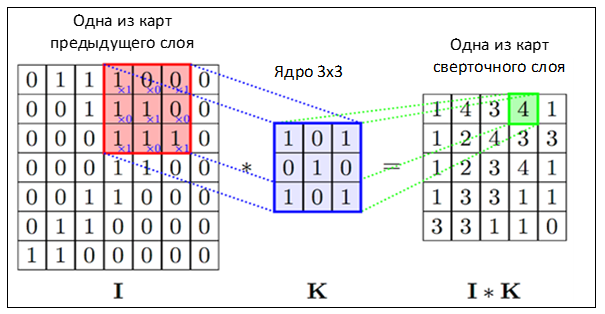
Ядро представляет из себя фильтр или окно, которое скользит по всей области предыдущей карты и находит определенные признаки объектов. Размер ядра обычно берут в пределах от 3х3 до 7х7. Если размер ядра маленький, то оно не сможет выделить какие-либо признаки, если слишком большое, то увеличивается количество связей между нейронами. Также размер ядра выбирается таким, чтобы размер карт сверточного слоя был четным, это позволяет не терять информацию при уменьшении размерности в подвыборочном слое, описанном ниже.

Ядро представляет собой систему разделяемых весов или синапсов, это одна 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 – наоборот, общие веса позволяет сократить число связей и позволить находить один и тот же признак по всей области изображения.

Изначально значения каждой карты сверточного слоя равны 0. Значения весов ядер задаются случайным образом в области от -0.5 до 0.5. Ядро скользит по предыдущей карте и производит операцию свертка, которая часто используется для обработки изображений, формула:



Неформально эту операцию можно описать следующим образом — окном размера ядра g проходим с заданным шагом (обычно 1) все изображение f, на каждом шаге поэлементно умножаем содержимое окна на ядро g, результат суммируется и записывается в матрицу результата, как на рис. 25 [21].

  
Рис. 25. Операция свертки и получение значений сверточной карты

Подвыборочный слой также, как и сверточный имеет карты, но их количество совпадает с предыдущим (сверточным) слоем, их 6. 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. К тому же фильтрация уже ненужных деталей помогает не переобучаться.

Последний из типов слоев это слой обычного многослойного персептрона. Цель слоя – классификация, моделирует сложную нелинейную функцию, оптимизируя которую, улучшается качество распознавания. Нейроны каждой карты предыдущего подвыборочного слоя связаны с одним нейроном скрытого слоя. Таким образом число нейронов скрытого слоя равно числу карт подвыборочного слоя, но связи могут быть не обязательно такими.

Выходной слой связан со всеми нейронами предыдущего слоя. Количество нейронов соответствует количеству распознаваемых классов.

* 1. Процедура предварительной обработки входных данных для формирования обучающей выборки

Предварительный набор данных (DataSet) представляет из себя набор из 14 видеороликов разного качества, снятых в разное время суток, на которых наблюдается рабочий процесс на шихтовом дворе. Все видео-материалы представлены в черно-белом виде, с разрешением 640х480 пикселей.

Для обучения сверточной нейронной сети необходимо «разрезать» видео-материалы на отдельные кадры, на каждом из которых нужно обозначить сцепку, в виде координаты центра масс x;y, высоты и ширины выделенной области. Все полученные значения необходимо оформить в виде фото.jpg + \*.txt файла, в относительных координатах, пример, приведен в табл.2.

Таблица 2

Пример содержимого разметочного файла

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 | 0,456 | 0,456 | 0,789 | 0,455 |
| 1 | 0,125 | 0,641 | 0,777 | 0,741 |

Относительное значение получаются в результате расчета по формулам (1)-(4):

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |
|  | (3) |
|  | (4) |

где – абсолютное значение координаты X;

– абсолютное значение координаты Y;

– абсолютное значение ширины выделенной области;

– абсолютное значение высоты выделенной области;

– ширина изображения;

– высота изображения.

Для автоматического создания разметочных файлов можно использовать готовое решение, например, Metaeditor, BBox Label Tool, Cutter.

Например, в программе Cutter, необходимо выбрать видео-файл. После загрузки выбрать нужный кадр, нажать кнопку «Добавить образец», для появления области в виде прямоугольника, которым необходимо выделить посторонний объект (рис.26).

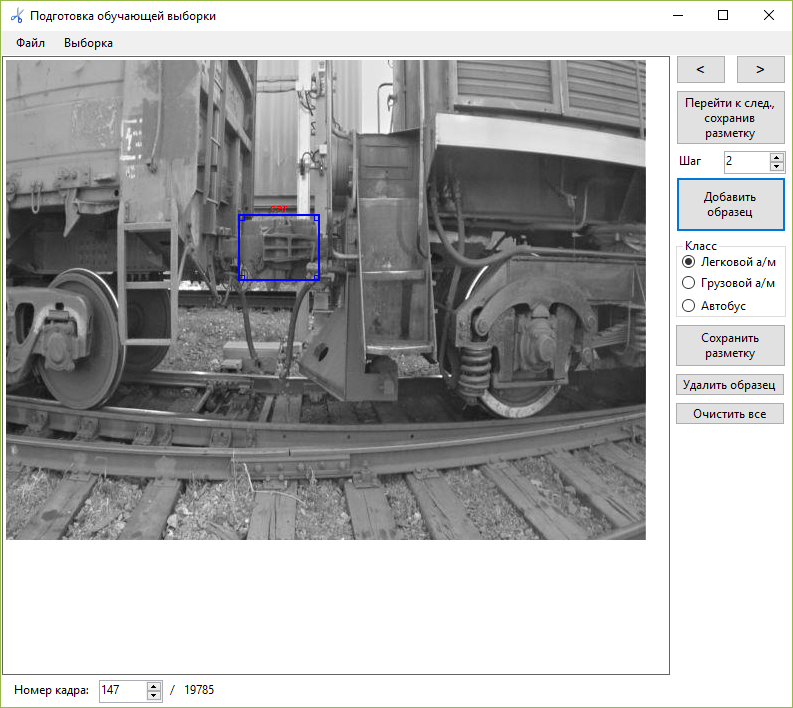


Рис.26. Программа Cutter

Для формирования .txt файл для каждого кадра по шаблону, описанному ранее, необходимо нажать кнопку «Сохранить разметку».

После подготовки файлов с разметкой, необходимо подготовить 3 файла, описанные в Прил.1, для обучения с помощью готового решения - Darknet, алгоритм которого описан в следующем пункте.

* 1. Алгоритм обучения сверточной нейронной сети

Для обучения используемой нейронной сети был использован алгоритм обратного распространения ошибки (backpropagation) [13]. Этот метод обучения многослойной нейронной сети называется обобщенным дельта-правилом. Метод был предложен в 1986 г. Румельхартом, Макклеландом и Вильямсом. Это ознаменовало возрождение интереса к нейронным сетям, который стал угасать в начале 70-х годов. Данный алгоритм является первым и основным практически применимым для обучения многослойных нейронных сетей [2].

Для выходного слоя корректировка весов интуитивна понятна, но для скрытых слоев долгое время не было известно алгоритма. Веса скрытого нейрона должны изменяться прямо пропорционально ошибке тех нейронов, с которыми данный нейрон связан [14]. Вот почему обратное распространение этих ошибок через сеть позволяет корректно настраивать веса связей между всеми слоями. В этом случае величина функции ошибки уменьшается и сеть обучается.

Основные соотношения метода обратного распространения ошибки получены при следующих обозначениях [8]:

– величина функции ошибки для образа ;

– желаемый выход нейрона для образа;

- активированный выход нейрона для образа ;

- взвешенная сумма выходов связанных нейронов предыдущего слоя на вес связи, по-другому еще обозначается не активированное состояние нейрона для образа;

– вес связи между и нейронами.

Величина ошибки определяется по формуле (5) среднеквадратичная ошибка:

|  |  |
| --- | --- |
|  | (5) |

где – величина функции ошибки для образа ;

– желаемый выход нейрона для образа;

- активированный выход нейрона для образа .

Не активированное состояние каждого нейрона для образа записывается в виде взвешенной суммы по формуле (6):

|  |  |
| --- | --- |
|  | (6) |

где – взвешенная сумма выходов связанных нейронов предыдущего слоя на вес связи, по-другому еще обозначается как не активированное состояние нейрона для образа ;

– вес связи между и нейронами;

– активированный выход нейрона для образа .

Выход каждого нейрона является значением активационной функции , которая переводит нейрон в активированное состояние. В качестве функции активации может использоваться любая непрерывно дифференцируемая монотонная функция. Активированное состояние нейрона вычисляется по формуле (7):

|  |  |
| --- | --- |
|  | (7) |

где - активированный выход нейрона для образа ;

– функция активности;

– не активированное состояние нейрона для образа .

В качестве метода минимизации ошибки используется метод градиентного спуска, суть этого метода сводится к поиску минимума (или максимума) функции за счет движения вдоль вектора градиента. Для поиска минимума движение должно быть осуществляться в направлении антиградиента. Метод градиентного спуска в соответствии с рис. 27.

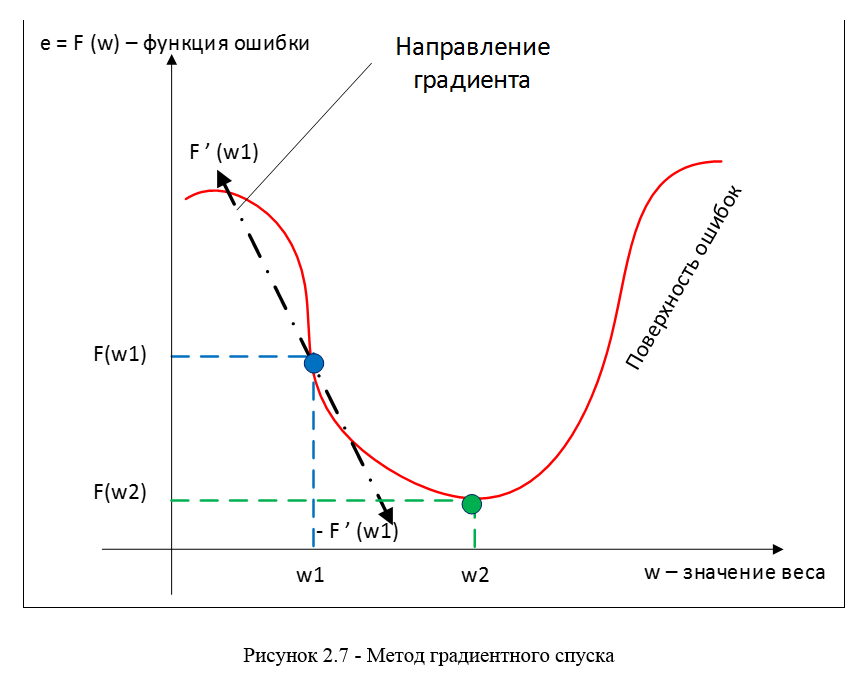


Рис.27. График функции ошибки

Градиент функции потери представляет из себя вектор частных производных, вычисляющийся по формуле (8):

|  |  |
| --- | --- |
|  | (8) |

где – градиент функции потери от матрицы весов;

– частная производная функции ошибки по весу нейрона;

– общее количество весов сети.

Производную функции ошибки по конкретному образу можно записать по правилу цепочки, формула (9):

|  |  |
| --- | --- |
|  | (9) |

где – значение производной функции ошибки по весу , между и нейронами;

– ошибка нейрона ;

– значение производной функции активации по её аргументу для нейрона , это часть достаточно просто можно вычислить;

– выход нейрона предыдущего слоя (по отношению к нейрону ).

Ошибка нейрона обычно записывается в виде символа (дельта). Для выходного слоя ошибка определена в явном виде, если взять производную от формулы (9), то получим минус , то есть разницу между желаемым и полученным выходом. Но как рассчитать ошибку для скрытых слоев? Для решения этой задачи, как раз и был придуман алгоритм обратного распространения ошибки. Суть его заключается в последовательном вычислении ошибок скрытых слоев с помощью значений ошибки выходного слоя, т.е. значения ошибки распространяются по сети в обратном направлении от выхода к входу.

Ошибка для скрытого слоя рассчитывается по формуле (10):

|  |  |
| --- | --- |
|  | (10) |

где – значение производной функции активации по ее аргументу для нейрона ;

– ошибка нейрона скрытого слоя;

– ошибка нейрона следующего слоя;

– вес связи между нейроном текущего (скрытого) слоя и нейроном выходного или тоже скрытого слоя.

Алгоритм распространения ошибки сводится к следующим этапам:

* прямое распространение сигнала по сети, вычисления состояния нейронов;
* вычисление значения ошибки для выходного слоя;
* обратное распространение: последовательно от конца к началу для всех скрытых слоев вычисляем по формуле (10);
* обновление весов сети на вычисленную ранее ошибки.

По окончанию обучения, мы получаем файл с набором весов, для каждой связи между и нейрона.

Далее приведем алгоритм детектирования и классификации состояния автосцепки, который будет использован в ПО на основе архитектуры AlexNet (DarkNet).

* 1. Алгоритм детектирования и классификации состояния автосцепки

Детектирование объектов – сложный алгоритм, включающий в себя операции по обработке входного файла (в частности, изображения) и, в зависимости от назначения, выдающий информацию об объекте в определенном формате [5].

Раньше, для детектирования объектов на изображении, применялся метод скользящего окна — когда прямоугольная область определённой ширины и высоты, перемещается («скользит») по изображению. Но теперь, с развитием свёрточных нейронных сетей и глубокого обучения, появились алгоритмы, которые на своём выходе дают не только класс объекта, но и ограничивающий объект прямоугольник: x,y,w,h. Одним из таких алгоритмов является семейство алгоритмов YOLO (You Only Look Once), которое подходит для работы в реальном времени, но имеет низкую точность на больших и маленьких объектах, средняя точность (mAP) — 63,4%. Однако в модификации YOLOv2 точность достигает точности Faster R-CNN. Также среди разработчиков имеет популярность алгоритм Alturos.Yolo, позволяющий более просто инсталлировать и использовать сам алгоритм в проектах. Этот алгоритм был использован в данном проекте и будет подробно рассмотрен далее.

Так как в этом подходе нейронная сеть предсказывает обрамляющие окна (bounding boxes) и вероятности классов, применяясь к полному изображению – нейросеть «видит всю картину целиком».

YOLO предполагает обнаружение объектов как единую задачу регрессии, которая начинается с пикселей входного изображения, а завершается вычислением координат обрамляющих прямоугольников и вероятностей принадлежности классам. При прогнозировании YOLO рассматривает изображение глобально. В отличие от методов скользящего окна и методов областных предложений, YOLO обрабатывает все изображение целиком во время обучения и тестирования, поэтому он неявно кодирует контекстную информацию о классах, а также их внешний вид, благодаря чему в этой модели ошибочное срабатывание детектора на фоне изображения менее вероятно, по сравнению с Fast R–CNN. При обучении на естественных изображениях и тестировании на художественных работах YOLO также превосходит DPM и R–CNN, с большим отрывом. Поскольку YOLO использует сильное обобщение, вероятность его ошибки при применении к новым входным данным также мала [4][5].

Главной задачей Alturos.Yolo стало повышение точности и качества локализации при сохранении своей скорости. Вместо растущей тенденции к укрупнению сети, для повышения производительности в Alturos.Yolo упрощается ее работа и вывод.

В Alturos.Yolo процесс «предсказывания координат» bounding box-а происходит с использованием опорных прямоугольников (англ. – anchor boxes) – стандартного набора прямоугольников с определенными значениями длины и ширины, которые получаются на этапе обучения с помощью метода кластеризации k–means. Каждой ячейке сетки сопоставляются по 5 таких anchor boxes разного размера, и вместо прямого предсказания bounding box сеть определяет отклонения от самого подходящего к объекту anchor box. Такие отклонения ограничены, что позволяет каждому anchor box фокусироваться на объекте определенной формы.

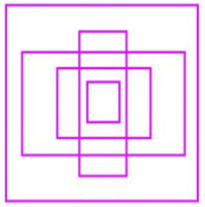


Рис.28.Опорные прямоугольники в Alturos.Yolo

Тоже происходит в случае, если центры разных объектов находятся в одной ячейке: каждому из них ставится соответствующий anchor box. Данное нововведение позволило YOLOv2 избавиться от полносвязных слоев, ранее отвечавших за предсказание bounding boxes, выводя вместо них результат, полученный от сверточных слоев, т.е YOLOv2 перешла на полностью сверточную нейронную сеть (FCN).

Также в Alturos.Yolo предсказания классов прикреплены к каждому bounding box, а не к ячейке, как это было в «классической» YOLO. Из–за введения anchor box и добавления новых классов детекции размеры векторов вероятности изменились на 1x425.

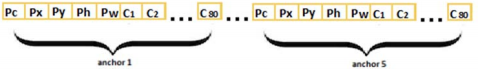


Рис.29. Вектора вероятности в Alturos.Yolo

В Alturos.Yolo вектор состоит из 5 разделов, по одному для каждого из anchor box. Параметры границ обнаруженных объектов вписываются в компоненты соответствующих anchor boxex. Для тех anchor boxes, для которых не нашлось подходящего объекта с центром в данной ячейке изображения компоненты Рс = 0. Alturos.Yolo также может работать с различными разрешениями для достижения компромисса между скоростью и точностью. Несмотря на разные входные разрешения обученная модель остается той же самой, без изменения весов.

Так же в данной модификации алгоритма был добавлен ряд улучшений, которые дают значительную прибавку в скорости и точности алгоритма, при этом точность достигла точности Faster R-CNN. Наиболее существенными улучшениями являются:

* Нормализация партии (Batch Normalization) дает прибавку точности в 2%. При обучении изображения подаются в нейросеть пачками, соответственно значения весов обновляются после обработки пачки. Нормализация партии приводит к существенному улучшению сходимости, устраняя необходимость в других формах регуляризации. Добавив нормализацию партии на всех сверточных уровнях в YOLO, получено более 2% улучшения в mAP. Нормализация партии также помогает упорядочить модель;
* High Resolution Classifier. Осуществлена точная настройка результирующей сети для обнаружения. Использован классификатор, обученный на наборе данных ImageNet. Эта сеть классификации с высоким разрешением дает прирост почти на 4% mAP;
* Dimension Clusters + Direct location prediction. Использование размерных кластеров наряду с прямым прогнозированием расположения центра обрамляющего окна улучшает YOLO почти на 5%. Таким образом точность YOLO c 63,4% поднялась до 76,8 %.

Описание алгоритма

На выходе нейросеть дает тензор размера 7×7×30, который содержит всю информацию, необходимую для нанесения на изображение обрамляющих окон.

1. На изображение наносится сетка 7×7. Если центр объекта попадает в ячейку сетки, эта ячейка сетки отвечает за обнаружение этого объекта.
2. Каждой клетке этой сетки соответствует вектор размера (4 + 1) × B + C, где:
3. каждое обрамляющее окно (bbox — bounding box) характеризуется пятью показателями:

* (x, y) — координаты центра соответствующей ячейки сетки;
* w — ширина обрамляющего окна относительно всего изображения;

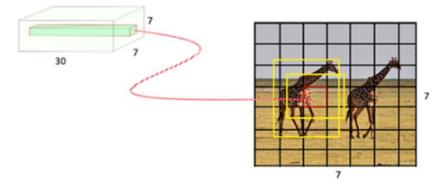


Рис. 30. Нанесенная сетка на изображение

* h — высота обрамляющего окна относительно всего изображения.
* некоторая вероятность того, что окно правильно нашло объект:

P r(object) ∗ IoUtruth predicted

Здесь ground-truth bboxes — размеченные вручную обрамляющие окна тестирующего набора данных, predicted bounding boxes — предсказанные моделью окна;

1. B — число обрамляющих окон для каждой ячейки сетки, B = 2;
2. C — число классов, которые могут быть предсказаны. В каждом из этих элементов вектора хранятся вероятности того, что внутри ячейки сетки лежит объект конкретного класса, без привязки к конкретному обрамляющему окну.
3. Таким образом, каждая ячейка сетки предсказывает B ограничивающих окон и доверительные оценки (confidence scores) для них. Теперь перемножим эти оценки каждого окна с вероятностями классов соответствующей ячейки сетки. Получим набор векторов размера C в количестве 2 × 7 × 7.
4. Для финального нанесения на изображение обрамляющих окон с полученным набором производятся следующие действия:

* рассматриваются значения набора векторов по классам;
* обнуляются не удовлетворяющие пороговому значению значения;
* сортируются по убыванию;
* применяется алгоритм Non-maximal suppressinon, который оставит наиболее точное из пересекающихся окон;
* на изображение наносятся только те окна, в векторе которых есть ненулевые значения, при этом окну присваивается класс с наибольшим значением.
  1. Адаптация и тестирование математической модели

Основной концепт реализации решения задачи заключается в обучении нейронной сети на текстуре сыпучего материала, для дальнейшего поиска посторонних предметов.

Для получения весов YOLOv2, необходимо провести обучение на подготовленной выборке – фотографиях сыпучего материала и файла разметки.

С сайта GitHub был склонирован репозиторий AlexeyAB / Darknet, который позволяет провести обучение и тестирование нейронной сети на нужной выборке. По итогу мы получим файл весов для YOLOv2, которые можем использовать в собственном ПО.

Для начала, необходимо скомпилировать darknet.exe, используя Visual Studio 2019 Community. Так как у нас CUDA 10.0, cuDNN 7.4 и OpenCV 3.x, открываем build\darknet\darknet.sln, устанавливаем компилирование кода в Release и для x64 битных систем (рис.21).

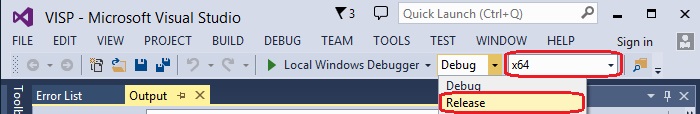


Рис. 21. Настройка компиляции

Также добавим системную переменную Windows CUDNN (рис.22):

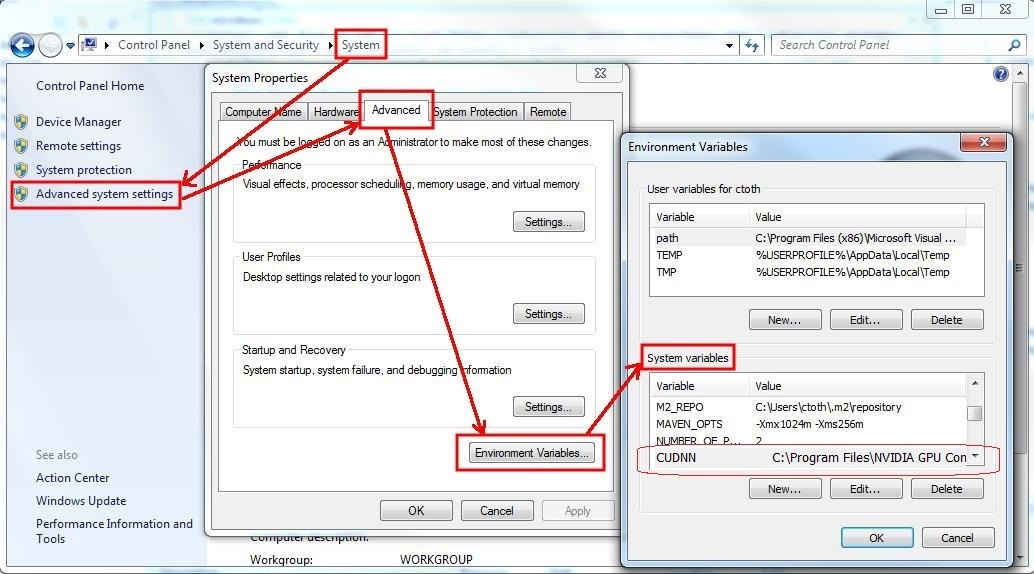


Рис. 22. Новая системная переменная

После успешной компиляции необходимо файлы OpenCV opencv\_world320.dll и opencv\_ffmpeg320\_64.dll скопировать в каталог, куда был скомпилирован darknet.exe. Туда же необходимо поместить файл cudnn64\_7.dll, для использования видеокарты в обучения нейронной сети.

Следующим шагом является подготовка cfg файла для Yolo2. В склонированном репозитории есть пример, который можно использовать, но необходимо исправить 2 строки:

1. Строка 241 – количество классов. Так как обучение нейронной сети будет проводиться на материал, то класс будет один.
2. Строка 234 – количество фильтров, рассчитанных по формуле (classes + 5) \* 5 = 30.

Для работы процесса обучения необходимо подготовить следующие файлы:

* Train\_trash\_severstal.data – файл, содержащий количество классов и пути к другим файлам.
* Train.list – пути к фото, которые были выбраны как обучающие (рис.23).
* Test.list – пути к фото, которые были выбраны как тестовые (рис.24).
* Label.list – список классов.

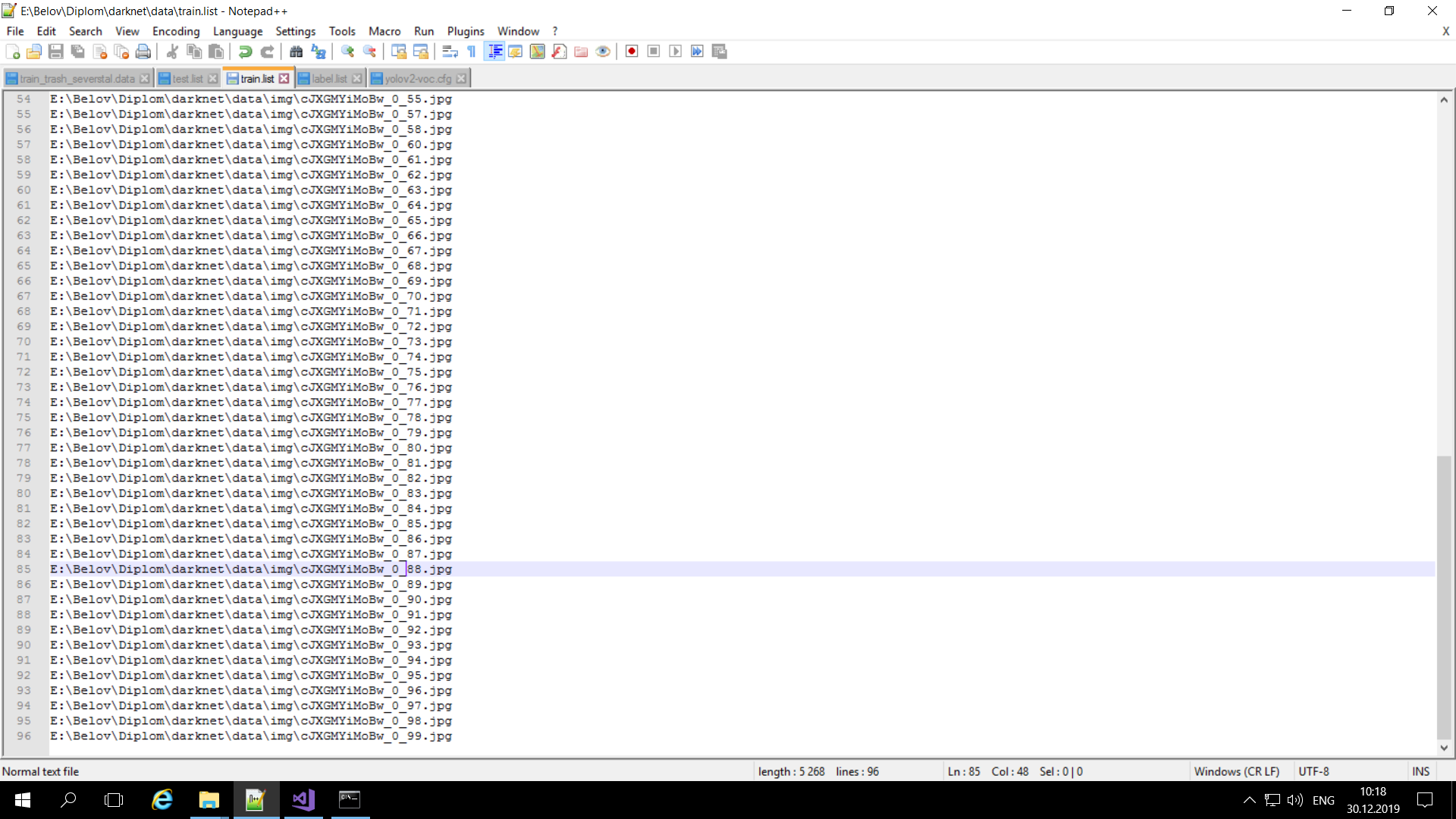


Рис. 23. Пример файла Train.list

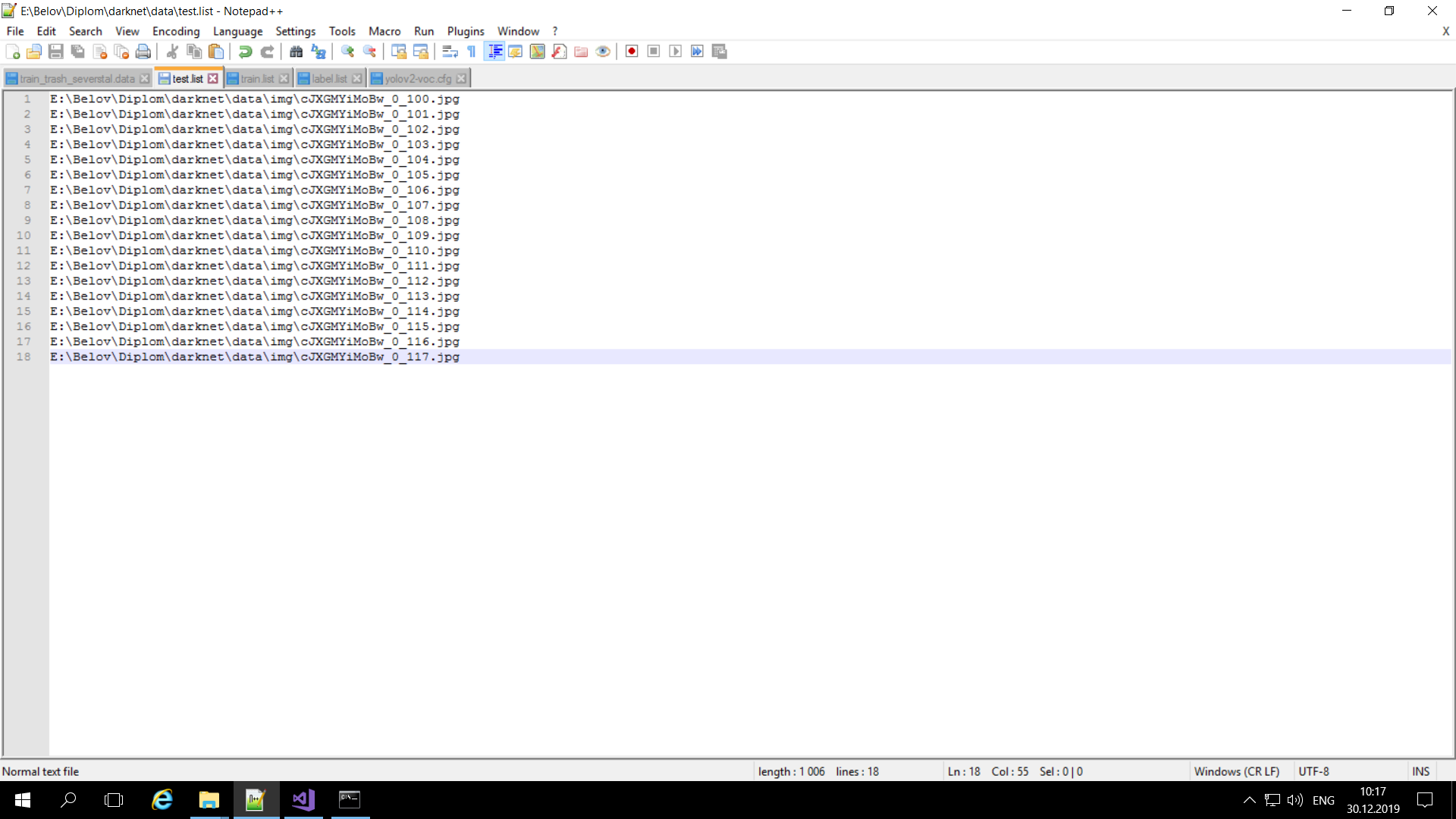


Рис. 24. Пример файла Test.list

Процесс обучения запускается посредством команды «darknet.exe detector train cfg\train\_trash\_severstal.data cfg\yolov2-voc.cfg cfg\darknet19\_448.conv.23 Pause» в cmd.exe (либо через \*.bat файл), где

* cfg\train\_trash\_severstal.data – файл, содержащий количество классов и пути к другим файлам (train.list, test.list, label.list).
* cfg\yolov2-voc.cfg – файл конфигурации нейронной сети.
* cfg\darknet19\_448.conv.23 – предварительные веса для нейронной сети.

По окончанию процесса обучения получаем следующий результат, представленный на рис.25 - 26.

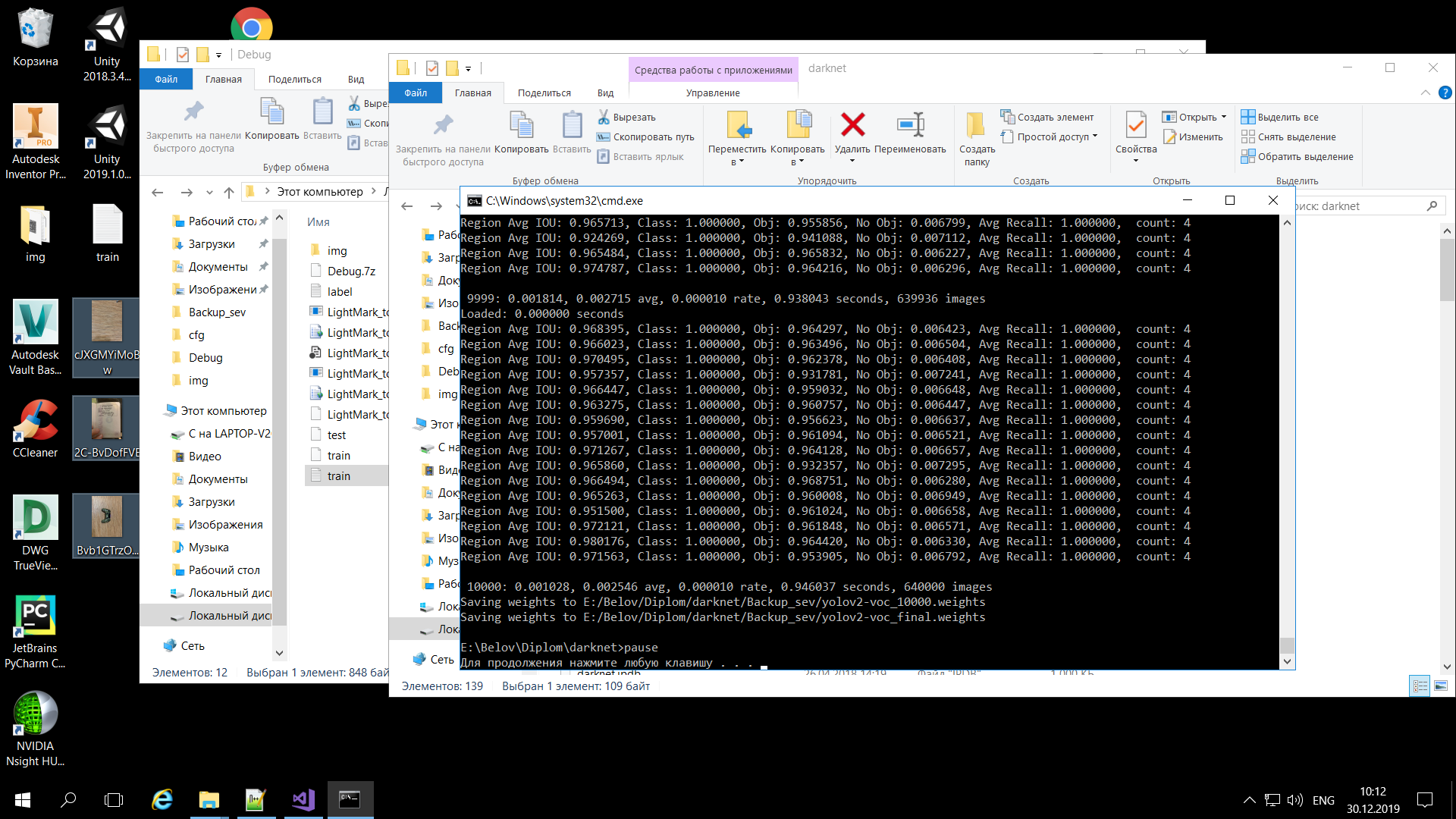


Рис. 25. Пример окончания обучения YOLO

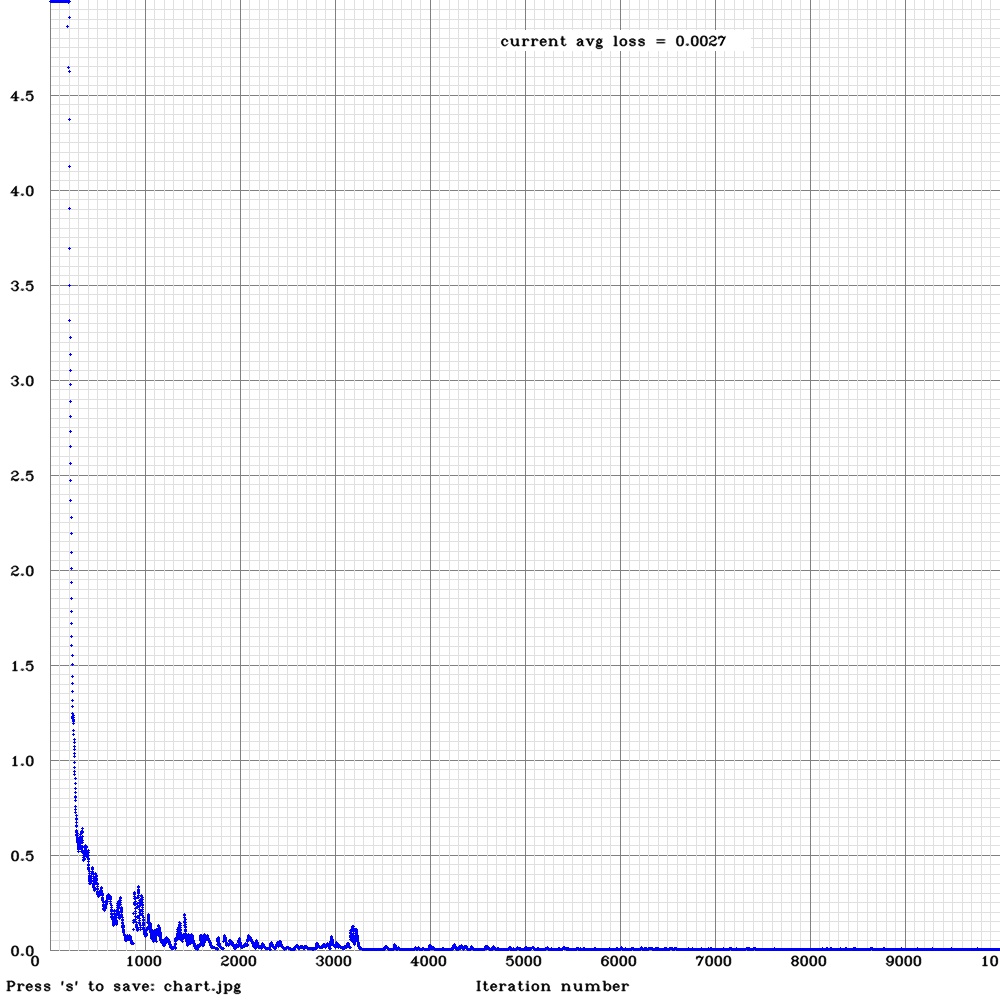


Рис. 26. Пример графика ошибки при обучении YOLO

Для тестирования запустим cmd команду:

* На фото - darknet.exe detector test cfg/coco.data cfg/yolov3.cfg yolov3.weights -thresh 0.25
* На видео – darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -ext\_output test.mp4

1. Проектирование системы распознавания посторонних предметов в сыпучих материалах
   1. Концептуальное моделирование проектируемой системы

Концептуальная модель – это систематизированное содержательное описание моделируемой системы (или проблемной ситуации) на неформальном языке. Неформализованное описание разрабатываемой имитационной модели включает определение основных элементов моделируемой системы, их характеристики и взаимодействие между элементами на собственном языке. При этом могут использоваться таблицы, графики, диаграммы и т.д. Неформализованное описание модели необходимо как самим разработчикам (при проверке адекватности модели, ее модификации и т.д.), так и для взаимопонимания со специалистами других профилей.

Модель диаграммы вариантов использования с последующим описанием представлена на рис.31.

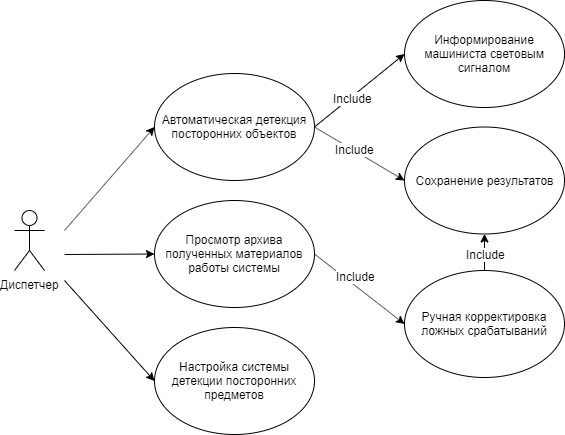


Рис. 31. Диаграмма вариантов использования

Таблица 3

Описание варианта использования «Автоматическая детекция посторонних объектов»

|  |  |
| --- | --- |
| Вариант использования | Автоматическая детекция посторонних объектов |
| 1 | 2 |
| Краткое описание | вариант использования, при котором сотрудник производит наблюдение в реальном времени за работоспособностью системы |
| Цель | Распознавание посторонних объектов |
| Тип | Основной |

Таблица 3. Продолжение

|  |  |
| --- | --- |
| 1 | 2 |
| Ссылки на другие варианты | * Информирование машиниста световым сигналом * Сохранение результатов |
| Исключения | * Нет подключения к БД * Нет подключения к IP-камере * Нет данных для загрузки нейронной сети |

Данный вариант использования начинает выполняться, когда пользователь запускает приложение.

Таблица 4

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1.  запускает приложения | 2. Система производит загрузку пользовательских настроек, нейронной сети, формирует пользовательский интерфейс. |
| 3. выбор видеопоток, если он не задан по умолчанию в настройках | 4. Система воспроизводит выбранный видеопоток, производя детекцию посторонних предметов. |

Таблица 5

Описание варианта использования «Информирование машиниста световым сигналом»

|  |  |
| --- | --- |
| Вариант использования | Информирование машиниста световым сигналом |
| Краткое описание | Это вариант использования, при котором система, найдя посторонний предмет в сыпучем материале, подаёт сигнал на уличный проблесковый маячок, тем самым предупреждаю машиниста бульдозера о наличии постороннего предмета, который необходимо извлечь. |
| Цель | Информирование машиниста |
| Тип | Основной |
| Ссылки на другие варианты | - |
| Исключения | - |

Таблица 6

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Запуск обработки видеопотока | 2. Система находит посторонний предмет и уведомляет диспетчера об этом соответствующим сообщением. |

Таблица 7

Описание варианта использования «Сохранение результатов»

|  |  |
| --- | --- |
| Вариант использования | Сохранение результатов |
| Краткое описание | это вариант использования, при котором система, найдя посторонний предмет в сыпучем материале, сохраняет полученные данные в базу данных, тем самым формируя архив. Для дальнейшего обучения системы, данный вариант использования еще включает в себя сохранение промежуточных кадром (каждые N). |
| Цель | Сохранять информацию для дальнейшего анализа и принятия решений |
| Тип | Основной |
| Ссылки на другие варианты | - |
| Исключения | * Нет подключение к БД |

Таблица 8

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Запуск обработки видеопотока | 1. При обнаружении постороннего предмета система сохраняет информацию о нем в БД. |

Таблица 9

Альтернативный ход событий №1

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Запуск обработки видеопотока | 1. Система каждые N кадров/минут/часов (в зависимости от настройки) сохраняет кадр в БД, для дальнейшего анализа |

Таблица 10

Альтернативный ход событий №2

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Разметка кадра | 1. Система сохраняет полученную информацию о разметке для дальнейшего обучения |

Таблица 11

Описание варианта использования «Просмотр архива полученных материалов работы системы»

|  |  |
| --- | --- |
| Вариант использования | Просмотр архива полученных материалов работы системы |
| Краткое описание | вариант использования, при котором диспетчер может просмотреть архив данных за необходимы промежуток времени, делать выгрузку в виде отчетов, для дальнейшего анализа и принятия решений |
| Цель | Просмотр материала для анализа и принятия решения |
| Тип | Основной |
| Ссылки на другие варианты | Ручная корректировка ложных срабатываний |
| Исключения | * Нет подключения к БД |

Таблица 12

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Запуск архива | 1. Система отображает всю сохраненную в БД ранее информацию о найденных посторонних предметах |

Таблица 13

Описание варианта использования «Ручная корректировка ложных срабатываний»

|  |  |
| --- | --- |
| Вариант использования | Ручная корректировка ложных срабатываний |
| Краткое описание | вариант использования, при котором диспетчер может на промежуточных кадрах исправить решение, принятое системой в ходе ее работы, с дальнейшим сохранением. Это необходимо для до обучения системы. |
| Цель | Подготовить материал для обучения системы |
| Тип | Основной |
| Ссылки на другие варианты | Сохранение результатов |
| Исключения | * Нет подключения к БД |

Таблица 14

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1. Запуск архива | 1. Система отображает всю сохраненную в БД ранее информацию о найденных посторонних предметах |
| 1. Разметка постороннего объекта, который не был распознан | 1. Сохранение полученной информации для дальнейшего обучения нейронной сети |

Таблица 15

Описание варианта использования «Настройка системы детекции посторонних предметов»

|  |  |
| --- | --- |
| Вариант использования | Настройка системы детекции посторонних предметов |
| Краткое описание | вариант использования, при котором диспетчер настраивает систему: подключение к базе данных, выбор видеопотока с ip-камеры и т.п. |
| Цель | Подготовка программы к работе |
| Тип | Основной |
| Ссылки на другие варианты | - |
| Исключения | * Неверные данные для подключения к БД * Неверные данные для подключения к ip-камерам |

Таблица 16

Типичный ход событий

|  |  |
| --- | --- |
| Действие исполнителя | Отклик системы |
| 1.  Запуск настроек системы | 2. Система загружает пользовательские настройки и формирует интерфейс |
| 3. Пользователь вносит изменения в конфигурацию системы | 4. Система «изменяется» согласно выбранным изменениям и сохраняет на локальном диске набор пользовательских настроек |

Диаграмма классов (англ. Static Structure diagram) — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними.

Контекстная диаграмма классов представлена на рис.32.

В предметной области можно выделить такие объекты, которые являются отдельными сущностями, как:

* диспетчер – сотрудник, которые производит наблюдения за шихтовым двором, посредством системы детекции посторонних объектов;
* машинист бульдозера – наемный рабочий, который посредством бульдозера перенаправляет шихту в бункер, для дальнейшего транспортирования по ленте в конверторный цех;
* сыпучий материал (шихта) – смесь материалов, загруженных в плавильную печь для получения металла определённого состава;
* посторонний предмет – предметы, попавшие в сыпучий материал. Это могут быть: проволока, прут, профиль и т.д.;

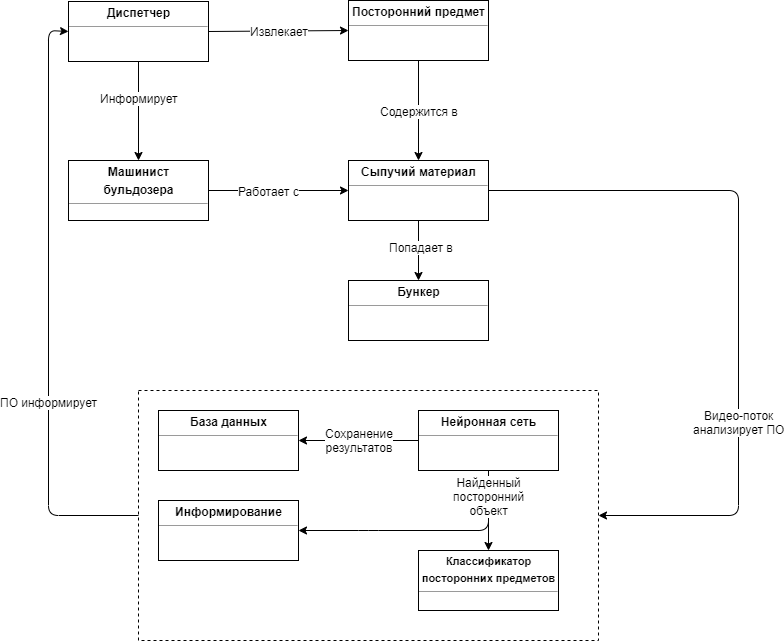


Рис. 32. Контекстная диаграмма классов

* бункер – специально оборудованное вместилище для сыпучих материалов.

В связи с тем, что предметная область декомпозируется на объекты, целесообразным подходом к проектированию ПО будет объектно-ориентированный подход (ООП).

* 1. Проектирование информационного обеспечения

В момент разработки конечного продукта будет предусмотрен алгоритм создания архива найденных посторонних предметов, который позволит подготовить отчет, позволяющий выдвигать гипотезы о происхождении посторонних предметов на шихтовом дворе.

Для хранения подобной информации была разработана ERD-диаграмма (рис.33).

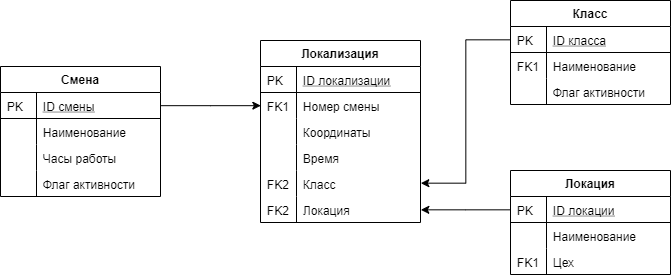


Рис. 33. ERD-диаграмма

DFD диаграмма представлена на рис.34-35.

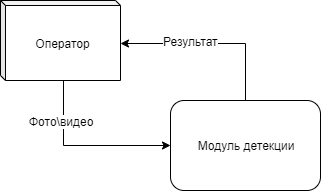


Рис. 34. DFD-диаграмма

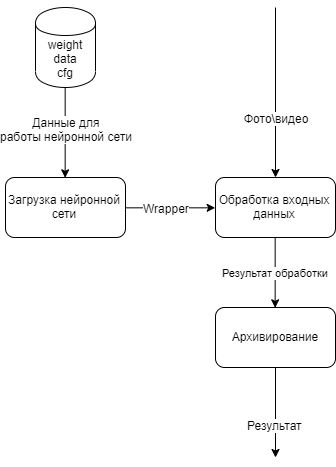


Рис. 35. Декомпозиция DFD диаграммы

* 1. Проектирование программного обеспечения

Процесс разработки ПО был выбран спиральный. Спиральная модель (рис.36) представляет шаблон процесса разработки ПО, который сочетает идеи итеративной и каскадной моделей. Суть ее в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

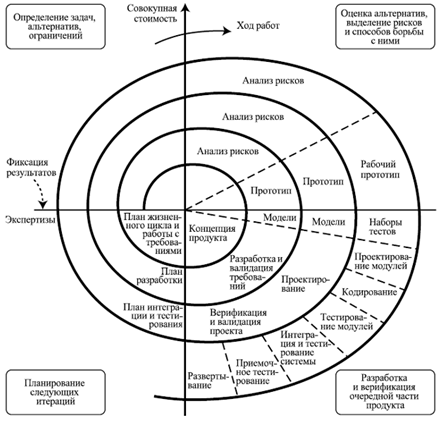


Рис. 36. Диаграмма спиральной модели

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд. Прототип, удовлетворяющий всем требованиям – готов к релизу.

Главная особенность спиральной модели – концентрация на возможных рисках. Для их оценки даже выделена соответствующая стадия. Основные типы рисков, которые могут возникнуть в процессе разработки ПО:

— Нереалистичный бюджет и сроки;

— Дефицит специалистов;

— Частые изменения требований;

— Чрезмерная оптимизация;

— Низкая производительность системы;

— Несоответствие уровня квалификации специалистов разных отделов.

Плюсы и минусы спиральной модели:

+ улучшенный анализ рисков;

+ хорошая документация процесса разработки;

+ гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;

+ раннее создание рабочих прототипов.

— может быть достаточно дорогой в использовании;

— управление рисками требует привлечения высококлассных специалистов;

— успех процесса в большой степени зависит от стадии анализа рисков;

— не подходит для небольших проектов.

Средой разработки выбрана Visual Studio 2019 Community.

Языком разработкой – C# с использованием технологии WPF, для построения пользовательского интерфейса.

Диаграмма пакетов

Независимо от типа проектируемого приложения и того, имеется ли у него пользовательский интерфейс или оно является сервисным приложением, которое просто предоставляет сервисы, его структуру можно разложить на логические группы программных компонентов. Эти логические группы называются слоями. Слои помогают разделить разные типы задач, осуществляемые этими компонентами, что упрощает создание дизайна, поддерживающего возможность повторного использования компонентов.

Каждый логический слой включает ряд отдельных типов компонентов, сгруппированных в подслои, каждый из подслоев выполняет определенный тип задач. Разделение приложения на слои, выполняющие разные роли и функции, помогает максимально повысить удобство и простоту обслуживания кода, оптимизировать работу приложения при различных схемах развертывания и обеспечивает четкое разграничение областей применения определенной технологии или принятия определенных проектных решений.

Типовой трехслойный дизайн слоев можно представить в виде диаграммы пакетов представлен на рис.37.

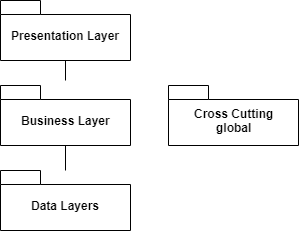


Рис.37. Диаграмма слоев

Рассмотрим каждый слой детальнее:

* Presentation Layer - слой представления. Данный слой содержит ориентированную на пользователя функциональность, которая отвечает за реализацию взаимодействием пользователя с системой, и, как правило, включает компоненты, обеспечивающие общую связь с основной бизнес-логикой, инкапсулириванной в бизнес-слое;
* Business Layer - бизнес-слой. Этот слой реализует основную функциональность системы и инкапсулирует связанную с ней бизнес-логику. Обычно он состоит из компонентов, некоторые из которых предоставляют интерфейсы сервисов, доступные для использования другими участниками взаимодействия.
* Слой доступа к данным. Этот слой обеспечивает доступ к данным, хранящимся в рамках системы, и данным, предоставляемым другими сетевыми системами. Доступ может осуществляться через сервисы. Слой данных предоставляет универсальные интерфейсы, которые могут использоваться компонентами бизнес слоя.

Рассмотрим Presentation Layer, используя паттерн проектирования MVVM.

Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model. Ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft.

Используется для разделения модели и её представления, что необходимо для их изменения отдельно друг от друга. Например, разработчик задаёт логику работы с данными, а дизайнер работает с пользовательским интерфейсом.

Cross Through Layer

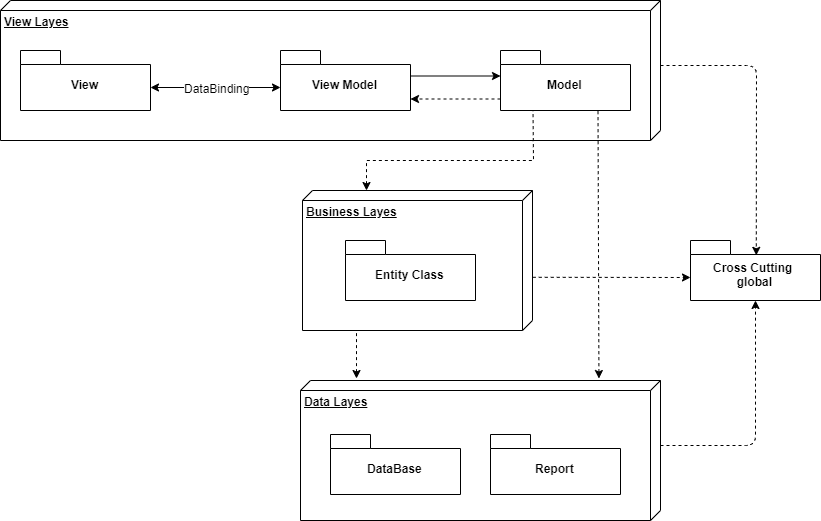


Рис.38. Детальная диаграмма пакетов

Каждый слой имеет ряд пакетов:

* View layer:
  + - View – графический интерфейс (окна, списки, кнопки и т. п.). Выступает подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью Представления. В случае, если в Модели Представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновлённое значение свойства из Модели Представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью Представления.
    - View Model – с одной стороны, абстракция Представления, а с другой — обёртка данных из Модели, подлежащие связыванию. То есть, она содержит Модель, преобразованную к Представлению, а также команды, которыми может пользоваться Представление, чтобы влиять на Модель.
    - Model – представляет собой логику работы с данными и описание фундаментальных данных, необходимых для работы приложения.
* Business layer:
  + - Entity Class – представляет собой набор классов-сущностей, которые будут описывать дополнительные объекты предметной области.
* Data layer:
  + - DataBase – представляет собой набор классов, описывающих взаимодействие системы с БД.
    - Report – представляет собой набор классов, реализующих возможность вывода информации из системы в видео отчетов (Excel, Word).

Диаграмма классов

Детальная диаграмма классов

В пакете View содержатся файлы описывающие визуальные компоненты форм, формирующие пользовательский интерфейс.

В пакте View-Model содержатся классы, содержащие объекты и методы, команды, которые помогают поддерживать состояние пользовательского интерфейса (представление), манипулировать моделью в результате действий над представлением и инициировать события в самом представлении.

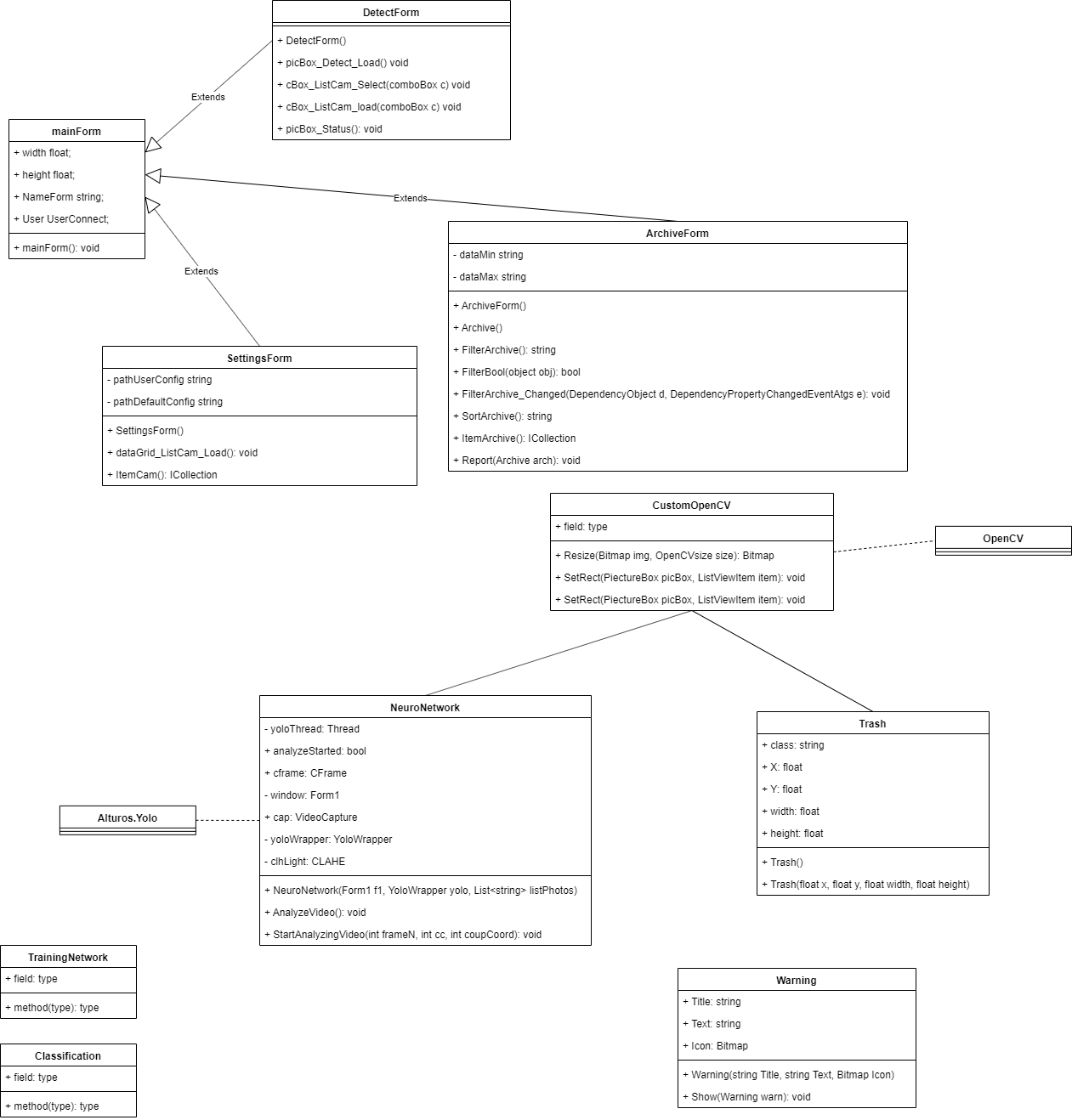


Рис.39. Детальная диаграмма классов пакета View Model

Класс mainForm

Класс mainForm описывает атрибуты и поведение компонентов родительской формы, которая будет содержать неизменные элементы интерфейса, такие как главное меню, системные кнопки «закрытия приложения», «развертывания на весь экран» и «сворачивания формы».

Описание класса mainForm представлено в табл.17-18.

Таблица 17

Атрибуты класса mainForm

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Тип данных | Модификатор доступа | Описание |
| width | float | public | Хранит ширину окна |
| height | float | public | Хранит длину окна |
| NameForm | string | public | Хранит имя формы |
| User | User | Public | Хранит информацию пользователя |

Таблица 18

Методы класса mainForm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование атрибута | Входные параметры | Тип возвращаемого значения | Модификатор доступа | Описание |
| mainForm | - | void | private | Инициализирует родительскую форму |

Класс DetectForm

Класс DetectForm описывает атрибуты и поведение компонентов дочерней формы, которая будет содержать неизменные элементы интерфейса, такие как окно вывода потокового видео выбранного файла или с IP-камеры, список которых также будет доступен в данной форме.

Описание класса mainForm представлено в табл.17-18.

Таблица 19

Методы класса DetectForm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование атрибута | Входные параметры | Тип возвращаемого значения | Модификатор доступа | Описание |
| DetectForm | - | void | private | Инициализирует родительскую форму |
| picBox\_Detect\_Load | - | void | private | Описывает работу компонента формы для вывода изображения на форму |
| cBox\_ListCam\_Select | comboBox c | void | private | Описывает работу компонента формы на вывод списка IP камер |
| cBox\_ListCam\_load | comboBox c | void | private | Описывает поиск и сохранение списка IP-камер |
| picBox\_Status | - | void | private | Описывает поведение статуса подключения к IP-камере |

Разработка интерфейса

В техническом задании заказчик указал ряд требований к пользовательскому интерфейсу, например, обязательная форма входа, логотоп компании, использование UI/UX.

В ходе проектирования программного обеспечения было принято решение строить логику интерфейса с использованием фреймов (frame). Данная технология позволит не создавать для каждого пункта меню отдельное окно, а выводить элементы интерфейса внутри главное фрейма.

С учётом этого были сформированы два окна:

1. Форма входа (рис.41) – содержит два поля ввода, для логина и пароля, checkbox для запоминания последнего введенного логина, а также кнопка входа.

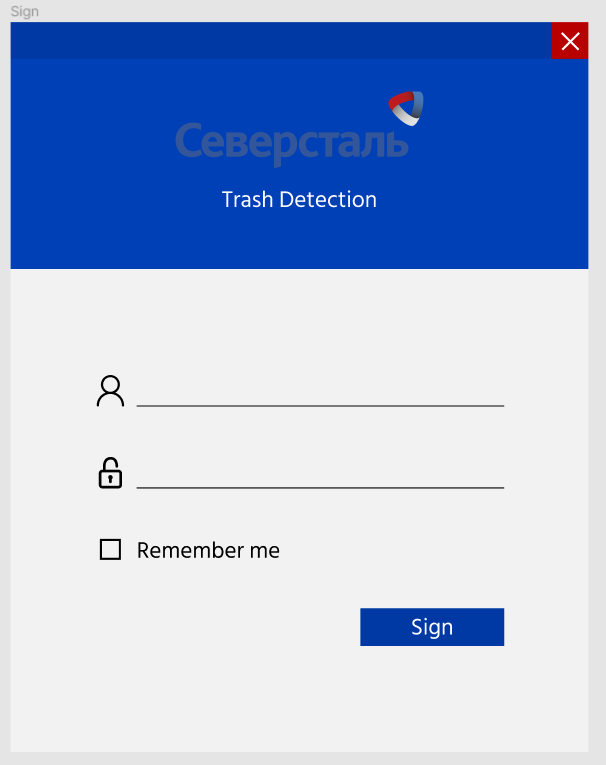


Рис.41. Форма входа

1. Главный фрейм (рис.42) – содержит стандартные кнопки манипуляции с окном («свернуть», «во весь экран», «закрыть»), главное меню, и пользователя, под которым был выполнен вход в приложение.

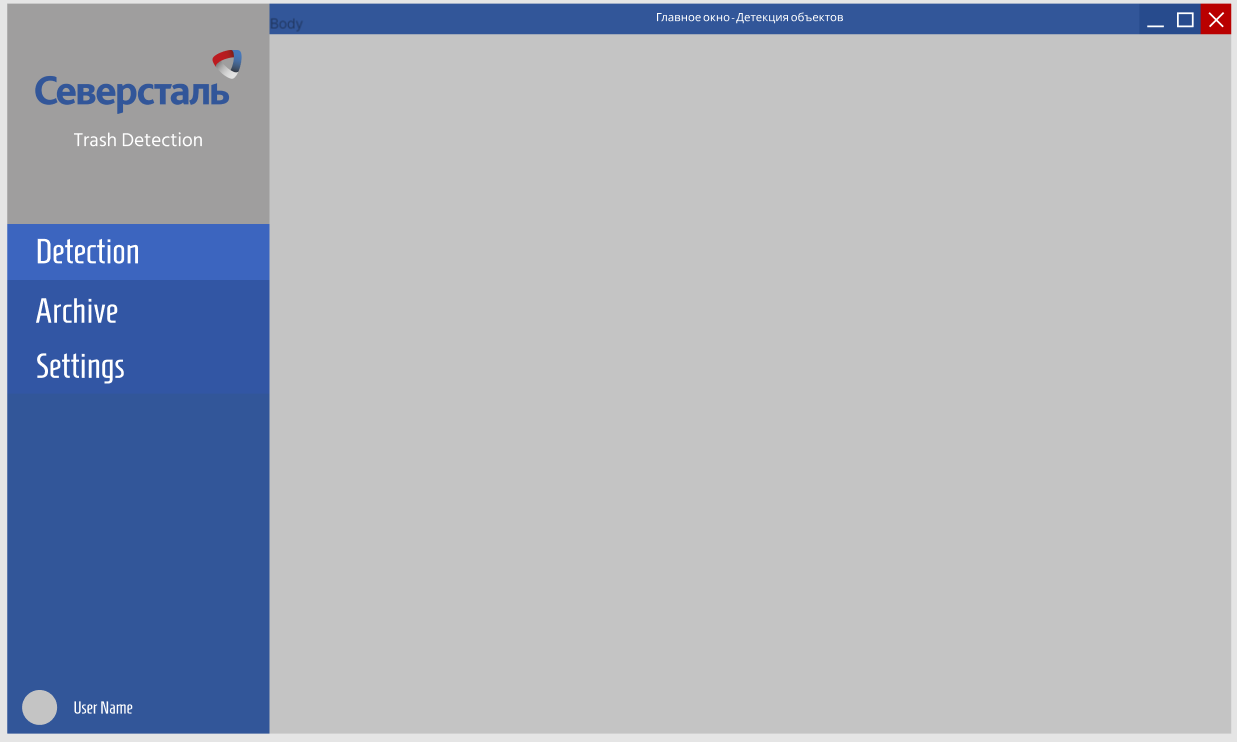


Рис. 42. Главное окно приложения

Так как компания Северсталь планирует данную разработку интегрировать в разные участки производства, в том числе и в других странах, то приложение будет разрабатываться с учетом возможности будущей мультилокализации.

# Список литературы

1. Goodfellow Ian. Deep Learning (Adaptive Computation and Machine Learning series). Cambridgе: The MIT Press, 2016.
2. Rashid Tariq. Make Your Own Neural Network. М.: Вильямс, 2018.
3. Валетов В.А. Аддитивные технологии (состояние и перспективы), Учебное пособие. СПб.: Университет ИТМО, 2015.
4. Джулли А. Библиотека Keras - инструмент глубокого обучения. М.: ДМК Пресс, 2017. 294 с.
5. Е.С.Борисов. Базовые методы обработки изображений, 2017.
6. Жерон О.Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники. М.: Вильямс, 2018.
7. Каменев С.В. Технологии аддитивного производства, Оренбург: Оренбургский государственный университет, 2017.
8. Лагунов Н.А. Выделение и распознавание объектов с использованием оптимизированного алгоритма селективного поиска и сверточной нейронной сети высокого порядка. Фундаментальные исследования. – 2015. – № 5 (часть 3). С. 511-516.
9. М. Н. Краснянский, А. Д. Обухов, Сравнительный анализ методов машинного обучения. Вестник ВГУ, серия: системный анализ и информационные технологии, 2018, № 3 – с.173.
10. Поляков А.Н. Основы быстрого прототипирования, Учебное пособие. Оренбург: ОГУ, 2014.
11. Рашид Тарик, Создаем нейронную сеть. М.: Вильямс, 2017.
12. Рудской А.И. Аддитивные технологии. Учебное пособие. СПб.: Изд-во Политехнического ун-та, 2016.
13. Саймон Хайкин. Нейронные сети. Полный курс. М.: Вильямс, 2016.
14. Хенрик Бринк, Джозеф Ричардс. Машинное обучение. СПб.: Питер, 2017.
15. Шишковский И.В. Основы аддитивных технологий высокого разрешения. СПб.: Питер, 2016.
16. Сайт компании ПАО «Северсталь». URL: https://severstal.com.
17. Свободная энциклопедия. URL: https://ru.wikipedia.org.
18. Метод опорных векторов. URL: http://statistica.ru/branches-maths/metod-opornykh-vektorov-supported-vector-machine-svm/.