**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 6](#_Toc516518752)

[**1.** **ОБЗОР ЛИТЕРАТУРЫ** 8](#_Toc516518753)

[**1.1** **Машинное обучение** 8](#_Toc516518754)

[**1.1.1 Алгоритмы машинного обучения** 9](#_Toc516518755)

[**1.2 Нейронные сети** 12](#_Toc516518756)

[**1.3 Сверточная нейронная сеть** 17](#_Toc516518757)

[**1.3.1 Слой свертки** 18](#_Toc516518758)

[**1.3.2 Функция активации** 19](#_Toc516518759)

[**1.3.3 Пулинг** 20](#_Toc516518760)

[**1.3.4 Полносвязная нейронная сеть** 21](#_Toc516518761)

[**1.4 Глубокое обучение** 21](#_Toc516518762)

[**1.5 Распознавание изображений** 25](#_Toc516518763)

[**1.5.1 Каскады Хаара** 29](#_Toc516518764)

[**2. ИСПОЛЬЗУЕМЫЕ СРЕДСТВА ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ** 32](#_Toc516518765)

[**2.1 Язык программирования Python** 32](#_Toc516518766)

[**2.1.1 История языка** 32](#_Toc516518767)

[**2.1.2 Синтаксис языка** 32](#_Toc516518768)

[**2.2 Интегрированная среда разработки PyCharm** 33](#_Toc516518769)

[**2.3.1 OpenCV и документация по библиотеке** 37](#_Toc516518770)

[**2.3.2 OpenCV на Python** 37](#_Toc516518771)

[**3. ПРОЦЕСС ИССЛЕДОВАНИЯ** 38](#_Toc516518772)

[**ЗАКЛЮЧЕНИЕ** 44](#_Toc516518773)

[**СПИСОК ЛИТЕРАТУРЫ** 45](#_Toc516518774)

[**ПРИЛОЖЕНИЕ 1** 47](#_Toc516518775)

[**ПРИЛОЖЕНИЕ 2** 48](#_Toc516518776)

[**ПРИЛОЖЕНИЕ 3** 50](#_Toc516518777)

[**ПРИЛОЖЕНИЕ 4** 51](#_Toc516518778)

[**ПРИЛОЖЕНИЕ 5** 54](#_Toc516518779)

# **ВВЕДЕНИЕ**

Темой данной дипломной работы является разработка системы распознавания изображений на основе аппарата искусственных нейронных сетей. Задача распознавания изображений является очень важной, так как возможность автоматического распознавания компьютером изображений приносит множество новых возможностей в развитии науки и техники, таких, как разработка систем поиска лиц и других объектов на фотографиях, контроля качества производимой продукции без участия человека, автоматического управления транспортом и множество других.

Что касается искусственных нейронных сетей, то в последние годы данный раздел машинного обучения получается все большее развитие в связи со значительным повышением вычислительных мощностей существующих компьютеров и повсеместном распространении применения графических карт для вычислений, что позволяет обучать нейронные сети гораздо большей глубины и сложной структуры чем раньше, которые, в свою очередь, показывают значительно лучшие результаты по сравнению с другими алгоритмами для многих задач, в особенности задачи распознавания изображений.

Так как задача распознавания изображений является очень обширной и в большинстве случаев требует отдельного подхода для различных типов изображений, то рассмотреть задачу распознавания изображений в целом в рамках одного исследования практически невозможно, поэтому будем рассматривать такую задачу, как обнаружение кисти руки на изображении.

Таким образом, основной целью данной дипломной работы была разработка системы распознавания изображений на основе искусственных нейронных сетей для изображений кисти руки. Для достижения данной цели были сформулированы следующие задачи:

1. Выполнение аналитического обзора литературы по теме искусственных нейронных сетей и их применения для задачи распознавания изображений.
2. Рассмотрение методов распознавания объекта на изображении.
3. Разработка прототипа системы распознавания изображений. Результатом выполнения данной задачи должна являться программа, позволяющая пользователю загрузить изображение и обнаружить на данном изображении необходимый объект.

В ходе проведенной работы все поставленные задачи были выполнены. Конкретные результаты будут описаны в основной части работы.

1. **ОБЗОР ЛИТЕРАТУРЫ**
   1. **Машинное обучение**

Технология машинного обучения на основе анализа данных берёт начало в 1950-х гг., когда начали разрабатывать первые программы для игры в шашки. За прошедшие десятилетия общий принцип не изменился. Зато благодаря взрывному росту вычислительных мощностей компьютеров многократно усложнились закономерности и прогнозы, создаваемые ими, и расширился круг проблем и задач, решаемых с использованием машинного обучения. [1]

Чтобы запустить процесс машинного обучения, для начала необходимо загрузить в компьютер некоторое количество исходных данных, на которых ему предстоит учиться обрабатывать запросы. После этого программа уже сможет самостоятельно распознавать объект на вновь загруженных картинках.

В наше время компьютеры активно применяются в тех сферах, которые всегда считались подвластными только людям. Конечно, некоторые технологии и наработки ещё только входят на рынок и далеко не идеальны, но это только начало пути. Машинное обучение, дающее возможность компьютеру самостоятельно совершенствовать свою работу, открывает им возможности для бесконечного развития. Благодаря машинному обучению ПК учатся распознавать на фотографиях и рисунках не только человека, но и пейзажи, предметы, текст и цифры. Что касается текста, то и здесь не обойтись без машинного обучения: функция проверки грамматики сейчас присутствует в любом текстовом редакторе и даже в телефонах. Причем учитывается не только написание слов, но и контекст, оттенки смысла и другие тонкие лингвистические аспекты.

Кроме того, теперь компьютер может общаться с человеком на естественном языке посредством голоса – слушать ваши реплики и давать ответы. Это виртуальные ассистенты – «Cortana», «Siri», «Google Now».

Наверняка вы не раз обращали внимание на то, что после поиска какого-либо товара в Интернете его реклама надолго поселяется в вашем браузере. В контекстной рекламе применяются весьма сложные и продвинутые алгоритмы персонализации, основанные на машинном обучении. Например, уже сейчас можно автоматически рассылать электронные письма, купоны, предложения и отображать рекомендации, подобранные под каждого клиента индивидуально. Это всё работает на стимулирование каждого пользователя к покупкам.

Очень востребованы системы машинного обучения, связанные с обработкой речи и текстов. В частности, это создание ботов для замены сотрудников в службах поддержки, которые оперативно отвечают на входящие запросы клиентов по заданным схемам. Сюда же относится расшифровка документации, которая постоянно необходима в юридической практике.

Выделяют два основных типа машинного обучения: обучение с учителем и обучение без учителя. Обучение с учителем предполагает, что алгоритму кроме самих исходных данных предоставляется некоторая дополнительная информация о них, которую он может в дальнейшем использовать для обучения. К числу наиболее популярных задач для обучения с учителем относятся задачи классификации и регрессии.

Обучение без учителя отличается от обучения с учителем тем, что алгоритму не предоставляется никакой дополнительной информации кроме самого набора исходных данных. Наиболее популярным примером задачи обучения без учителя является задача кластеризация.

# **1.1.1 Алгоритмы машинного обучения**

Существует множество алгоритмов машинного обучения, выделим здесь самые распространенные из них.

* Дерево принятия решений — средство поддержки принятия решений, которое использует древовидный граф или модель принятия решений, а также возможные последствия их работы, включая вероятность наступления события, затраты ресурсов и полезность. На рисунке 1 подано графическое представление структуры дерева.

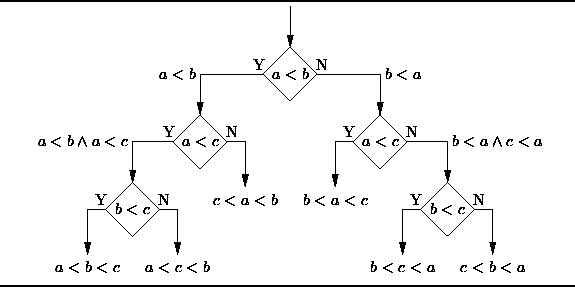
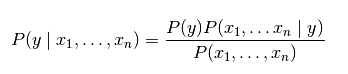


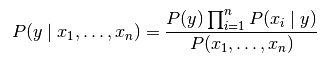
Рисунок 1

* Байесовский классификатор – как следует из названия, классификатор основан на теореме Байеса, которая записывается в форме



**нейронный сеть машинный обучение**

Идея классификатора в данном случае заключается в том, чтобы найти класс с максимальной апостериорной вероятностью при условии, что все параметры имеют те значения, которые они имеют для классифицируемого экземпляра. В общем случае данная задача предполагает предварительного знания очень большого количества условных вероятностей и, соответственно, огромного размера обучающей выборки и высокой сложности вычислений, поэтому на практике чаще всего применяется разновидность байесовского классификатора, называемая Наивным Байесовским классификатором, в которой предполагается, что все параметры независимы друг от друга, соответственно, формула принимает гораздо более простой вид и для ее использования требуется знать лишь небольшое количество условных вероятностей.



Хотя данное допущение обычно является далеким от реальности, наивный байесовский классификатор часто показывает неплохие результаты [2].

* Метод наименьших квадратов выступает для реализации линейной регрессии. Чаще всего она представляется в виде задачи подгонки прямой линии, проходящей через множество точек. Есть несколько вариантов ее осуществления, и метод наименьших квадратов — один из них. Можно нарисовать линию, а затем измерить расстояние по вертикали от каждой точки к линии и «перенести» эту сумму вверх. Необходимой линией будет та конструкция, где сумма расстояний будет минимальной. Иными словами, кривая проводится через точки, имеющие нормально распределенное отклонение от истинного значения. (Рисунок 2)

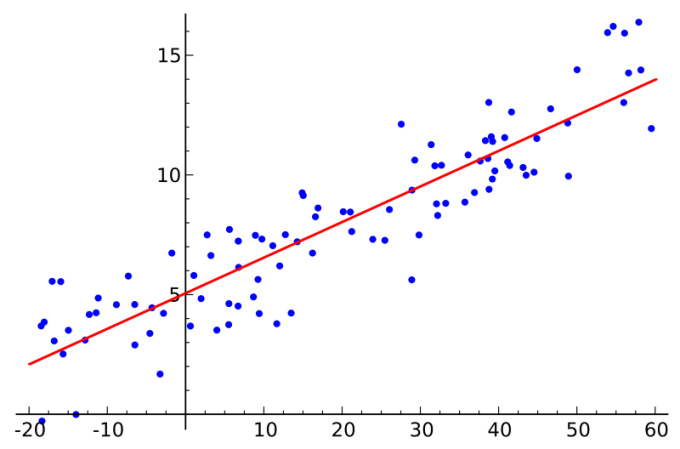


Рисунок 2

* Логистическая регрессия представляет собой мощный статистический способ прогнозирования вероятности возникновения некоторого события с одной или несколькими независимыми переменными. Логистическая регрессия определяет степень зависимости между категориальной зависимой и одной или несколькими независимыми переменными путем использования логистической функции, являющейся аккумулятивным логистическим распределением.
* Метод главных компонент — один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации.
* Анализ независимых компонент – статистический метод выявления скрытых факторов, которые лежат в основе множества случайных величин, сигналов и прочих измерений. Данный алгоритм непосредственно связан с методом главных компонент, но это гораздо более мощная техника, способная найти скрытые факторы источников.

# **1.2 Нейронные сети**

Искусственные нейронные сети (ИНС) являются достаточно известной и используемой моделью машинного обучения. Идея ИНС базируется на имитации нервной системы животных и людей.

Упрощенная модель нервной системы животных при этом представляется в виде системы клеток, каждая из которых имеет тело и ответвления двух типов: дендриты и аксоны. В определенный момент клетка получает сигналы от других клеток через дендриты и, если эти сигналы будут достаточной силы, возбуждается и передает это возбуждение другим клеткам, с которыми она связана, через аксоны. Таким образом сигнал (возбуждение) распространяется по всей нервной системе. Модель нейронных сетей устроена аналогичным образом. Нейронная сеть состоит из нейронов и направленных связей между ними, при этом каждая связь имеет некоторый вес. При этом часть нейронов является входными - на них поступают данные из внешней среды. Затем на каждом шаге нейрон получает сигнал от всех входных нейронов, вычисляет взвешенную сумму сигналов, применяет к ней некоторую функцию и передает результат на каждый из своих выходов. Также сеть имеет некоторое количество выходных нейронов, которые формируют результат работы сети.

Выделяется несколько основных типов архитектур нейронных сетей:

* Сеть прямого распространения (feed-forward network) – подразумевает, что нейроны и связи между ними образуют ациклический граф, где сигналы распространяются только в одном направлении. Именно такие сети являются наиболее популярными и широко изученными, и их обучение представляет наименьшие трудности.
* Рекуррентные нейронные сети (recurrent neural networks) – в таких сетях, в отличие от сетей прямого распространения, сигналы может передаваться в обоих направлениях, и могут поступать на один и тот же нейрон несколько раз в процессе обработки одного входного значения. Частной разновидностью рекуррентных нейронных сетей является, например, машина Больцмана. Основной трудностью в работе с такими сетями является их обучение, так как создать эффективный алгоритм для этого в общем случае является сложной задачей и до сих пор не имеет универсального решения.
* Самоорганизующиеся карты Кохонена – нейронная сеть, предназначенная в первую очередь для кластеризации и визуализации данных.

В истории развития нейронных сетей выделяют 3 основных периода подъема. Первые исследования в области искусственных нейронных сетей относятся к 40-м годам 20 века. В 1954 году У. Маккалок и У. Питтс опубликовали работу «Логическое исчисление идей, относящихся к нервной деятельности» [3], в которой были изложены основные принципы построения искусственных нейронных сетей. В 1949 году была опубликована книга Д.Хебба «Организация поведения», где автор рассмотрел теоретические основы обучения нейронных сетей и впервые сформулировал концепцию обучения нейронных сетей как настройку весов между нейронами. В 1954 году В.Кларк впервые осуществил попытку реализовать аналог сети Хебба с помощью компьютера. В 1958 году Ф. Росенблатт предложил модель персептрона, который представлял собой нейронную сеть с одним скрытым слоем [4]. Принципиальный вид персептрона Росенблатта представлен на рисунке 3.

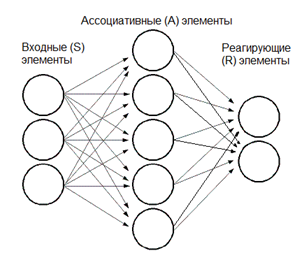


Рисунок 3. Персептрон Росенблатта

Данная модель обучалась с помощью метода коррекции ошибок, который заключался в том, что веса остаются неизменными до тех пор, пока выходное значение персептрона корректно, в случае же ошибки вес связи изменяется на 1 в сторону, обратную знаку произошедшей ошибки. Данный алгоритм, как было доказано Росенблаттом, всегда сходится. Используя такую модель, удалось создать компьютер, распознающий некоторые буквы латинского алфавита, что, несомненно, было большим успехом в то время.

Однако интерес к нейронным сетям существенно снизился после публикации М. Минским и С. Пейпертом книги «Персептроны» (“Perceptrons”) в 1971 году [5], где они описали существенные ограничения, которые имеет модель персептрона, в частности, невозможность представления функции исключающего или, а также указали на слишком высокие требования к требуемой вычислительной мощи компьютеров для обучения нейронных сетей. Так как данные ученые имели очень высокий авторитет в научном сообществе, нейронные сети были на некоторое время признаны малоперспективной технологией. Ситуация изменилась только после создания в 1974 году алгоритма обратного распространения ошибки.

Алгоритм обратного распространения ошибки (backpropagation algorithm) был предложен в 1974 году одновременно и независимо двумя учеными, П.Вербосом и А. Галушкиным [6, 7]. Данный алгоритм основан на методе градиентного спуска. Основная идея алгоритма заключается в распространении информации об ошибке от выходов сети к ее входам, то есть в обратном направлении по отношению к стандартному походу. При этом веса связей корректируются на основе дошедшей до них информации об ошибке. Основное требование, которое накладывает данный алгоритм – то, что функция активации нейронов должна быть дифференцируема, так как метод градиентного спуска, что неудивительно, вычисляется на основе градиента.

Алгоритм обратного распространения ошибки позволяет легко обучать сеть, имеющую несколько скрытых слоев, что позволяет обойти ограничения персептрона, блокирующие развитие данной отрасли ранее. С математической точки зрения данный алгоритм сводится к последовательному перемножению матриц – что является достаточно хорошо изученной и оптимизируемой задачей. Кроме того, данный алгоритм хорошо параллелизуется, что позволяет существенно ускорить время обучения сети. Все это совместно привело к новому расцвету нейронных сетей и множеству активных исследований в данном направлении.

Алгоритм backpropagation, в то же время, имеет ряд проблем. Так, использование градиентного спуска предполагает риск схождения к локальному минимуму. Другой важной проблемой является долгое время обучения алгоритма при наличии большого количества слоев, так как ошибка в процессе обратного распространения имеет свойство все сильнее уменьшаться при приближении к началу сети, соответственно, обучение начальных слоев сети будет происходить крайне медленно. Еще одним недостатком, свойственным нейронным сетям в целом, является сложность в интерпретации результатов их работы. Обученная модель нейронной сети представляет собой некоторое подобие черного ящика, на вход которой подается объект и на выходе получается прогноз, однако определить, какие признаки входного объекта при этом учитывались и какой из нейронов за что отвечает, обычно достаточно проблематично. Это делает нейронные сети во многом менее привлекательными по сравнению, например, с деревьями решений, в которых обученная модель сама по себе представляет некоторую квинтэссенцию знаний о рассматриваемой предметной области и исследователю легко понять, почему данный объект был отнесен к тому или иному классу.

Данные недостатки, в сочетании с тем, что, хотя нейронные сети и показывали неплохие результаты, эти результаты были сравнимы с результатами других классификаторов, например, набирающих популярность машин опорных векторов, при этом результаты последних были гораздо проще в интерпретации, а обучение требовало меньшего времени, привело к очередному спаду развития нейронных сетей.

Этот спад завершился только в 2000-х годах 21 века, когда появилось и стало распространяться понятие deep learning, или глубокого обучения. Возрождению нейронных сетей способствовало появление новых архитектур, таких, например, как сверточные сети, restricted bolzman machines, стековые автоэнкодеры и т.д., которые позволили добиться существенно более высоких результатов в таких сфера машинного обучения, как распознавание изображений и речи. Существенным фактором для их развития стало также появление и распространение мощных видеокарт, а также их применение для вычислительных задач. Видеокарты, отличаясь значительно большим количеством ядер по сравнения с процессором, пусть и меньшей мощности каждое, идеально подходят для задач обучения нейронных сетей. Это в сочетании с существенно возросшей в последнее время производительностью компьютеров в целом и распространением вычислительных кластеров позволило обучать существенно более сложные и глубокие архитектуры нейронных сетей, чем ранее.

# **1.3 Сверточная нейронная сеть**

Сверточная нейронная сеть (англ. convolutional neural network - CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном и нацеленная на эффективное распознавание изображений, входит в состав технологий глубокого обучения (англ. Deep learning) [8]. Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоев и субдискретизирующих слоев (слоёв подвыборки). Структура сети — однонаправленная, принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов — любая, по выбору исследователя. Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

В машинном обучении сверточная нейронная сеть - это такой тип искусственной нейронной сети прямого распространения, в котором отдельные нейроны заключены таким образом, что они реагируют на области зрительного поля, которые частично перекрываются. В частности, CNN широко применяются в распознавании объектов на изображении и видео, рекомендательных системах и обработке языка. Разработка сверточных нейронных сетей последовала за открытием зрительных механизмов в живых организмах.

Неокогнитрон, предшественник сверточных сетей, был представлен в работе 1980 года [9]. Их конструкция была усовершенствована 1998 года [10], обобщена в 2003 и того же года упрощена [11]. Знаменитая сеть LeNet-5 может успешно классифицировать цифры, и применяется для распознавания цифр в чеках. Однако при более сложных задачах широта размаха и глубина сети растут, и вычислительные ресурсы не позволяют находить решение. В 2006 вышло несколько публикаций, где описали эффективные пути тренировки сверточных нейронных сетей с большим количеством слоев [12]. С 2011 года они были уточнены и реализованы с впечатляющими результатами. В 2012 году удалось значительно усовершенствовать производительность в литературе для нескольких наборов изображений, включая базы данных MNIST, NORB, набором символов HWDB1.0 (китайские символы), набором данных CIFAR10 (набор из 60000 меченых изображений RGB 32 × 32) и набором данных ImageNet.

# **1.3.1 Слой свертки**

Является основном блоком сверточной нейронной сети. Слой свертки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам, суммируя результаты матричного произведения для каждого фрагмента. Весовые коэффициенты ядра свертки неизвестны и устанавливаются в процессе обучения. Операция наглядно продемонстрирована на рисунке 4:

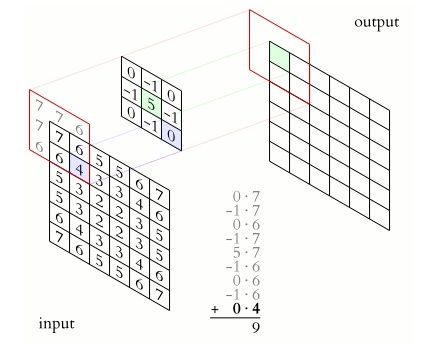


Рисунок 4

# **1.3.2 Функция активации**

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки (считают, что функция активации встроена в слой свёртки). Функция нелинейности может быть любой по выбору исследователя, традиционно для этого использовали функции типа гиперболического тангенса или сигмоиды . Однако в 2000х годах была предложена и исследована новая функция активации — ReLU (сокращение от англ. rectified linear unit), которая позволила существенно ускорить процесс обучения и одновременно упростить вычисления (за счёт простоты самой функции)[6], что означает блок линейной ректификации, вычисляющий функцию То есть по сути это операция отсечения отрицательной части скалярной величины. По состоянию на 2017 год эта функция и её модификации (Noisy ReLU, Leaky ReLU и другие) являются наиболее часто используемыми функциями активации в глубоких нейросетях, в частности, в свёрточных.

# **1.3.3 Пулинг**

Слой пулинга (иначе подвыборки, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения. Подвыборка интерпретируется так. Если на предыдущей операции свёртки 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. Слой пулинга, как правило, вставляется после слоя свёртки перед слоем следующей свёртки. Выглядит операция субдискретизации, как показано на рисунке 5.

 Рисунок 5

# **1.3.4 Полносвязная нейронная сеть**

После нескольких прохождений свёртки изображения и уплотнения с помощью пулинга система перестраивается от конкретной сетки пикселей с высоким разрешением к более абстрактным картам признаков, как правило на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. В конечном счете остаётся большой набор каналов, хранящих небольшое число данных, которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения.

Эти данные объединяются и передаются на полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью. В полном объеме архитектура нейронной сети выглядит, как показано на рисунке 6.

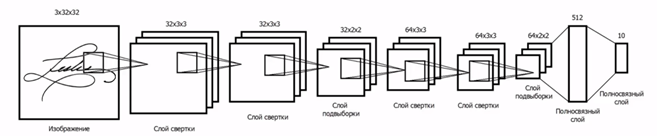


Рисунок 6

# **1.4 Глубокое обучение**

Одной из наиболее важных проблем, с которыми приходится сталкиваться при использовании алгоритмов машинного обучения, является проблема выбора правильных признаков, на основе которых производится обучение. Особенно значимой становится данная проблема при рассмотрении таких задач, как распознавание изображений, распознавание речи, обработка естественного языка и тому подобных, то есть тех, где отсутствует очевидный набор признаков, которые могут быть использованы для обучения. Обычно выбор набора признаков для обучения осуществляется самим исследователем путем некоторой аналитической работы, и именно выбранный набор признаков во многом определяет успешность работы алгоритма. Так, для задачи распознавания изображений в качестве таких признаков может служить преобладающий цвет в изображении, степень его изменения, наличие на изображении четких границ или что-то еще. Более подробно вопрос распознавания изображений и выбора правильных признаков для этого будет рассмотрен в соответствующей главе.

Однако такой подход имеет существенные недостатки. Во-первых, данный метод подразумевает существенный объем работы по выявлению признаков, причем эта работа осуществляется вручную исследователем и может требовать больших затрат времени. Во-вторых, выявление признаков, на основании которых можно получить качественный алгоритм, в данном случае становится во многом случайным, к тому же, таким образом маловероятно, что будут приняты во внимание признаки, которые могут оказывать важное влияние на внутреннюю структуру изображения, но при этом неочевидны для человека. Следуя из рассуждений, особенно привлекательной выглядит идея автоматического определения признаков, которые в дальнейшем могут быть использованы для работы алгоритмов машинного обучения. И именно такую возможность предоставляет использование подхода deep learning (глубокое обучение).

С точки зрения теории машинного обучения, deep learning является подмножеством так называемого representation learning. Основная концепция representation learning – как раз-таки автоматический поиск признаков, на основании которых в дальнейшем будет работать некоторый алгоритм, например, классификации.

С другой стороны, еще одна важная проблема, с которой приходится сталкиваться при использовании машинного обучения – это наличие факторов вариации, которые могут оказать существенное влияние на внешний вид исходных данных, однако при этом не имеют отношения к самой их сути, которую исследователь и пытается анализировать [13]. Так, в задаче распознавания изображений такими факторами могут являться угол, под которым предмет на изображении повернут к наблюдателю, время суток, освещение и т.д. Также в зависимости от точки зрения и погоды красная машина может иметь на фотографии различный оттенок и форму. Поэтому для подобных задач, например, идентификации предмета, изображенного на фотографии, выглядит разумным учитывать не конкретные низкоуровневые факты, такие как цвет определенного пикселя, а характеристики более высокого уровня абстракции, например, наличие колес. Однако очевидно, что определить на основе исходного изображения, присутствуют ли у него колеса – задача нетривиальная, и ее решение напрямую может быть весьма сложным. Кроме того, наличие колес – только одно из огромного множества возможных признаков, и определение их всех и составление алгоритмов для проверки изображения на наличие их выглядит не очень реалистичным. Именно здесь исследователи могут использовать все преимущества подхода deep learning. Deep learning основан на предоставлении исходного объекта в виде иерархической структуры признаков, таким образом, что каждый следующий уровень признаков строится на основе элементов предыдущего уровня. Так, если речь идет об изображениях, в качестве самого низшего уровня будут выступать исходные пиксели изображения, следующим уровнем будут отрезки, которые можно выделить среди этих пикселей, затем – углы и другие геометрические фигуры, в которые складываются отрезки. На следующем уровне их фигур образуются уже узнаваемые для человека объекты, например, колеса, и наконец, последний уровень иерархии отвечает за конкретные предметы на изображении, например, автомобиль.

Для реализации подхода deep learning в современной науке используются многослойные нейронные сети различных архитектур. Нейронные сети идеально подходят для решения задачи выявления из данных и построения иерархического множества признаков, так как, по сути, нейронная сеть представляет собой совокупность нейронов, каждый из которых активируется только если входные данные удовлетворяют определенным критериям – то есть представляет собой некоторый признак, при этом правила активации нейрона – то, что и определяет данный признак -обучаются автоматически. В то же время, нейронные сети в своем наиболее распространенном виде, сами по себе представляют иерархичную структуру, где каждый следующий слой нейронов использует в качестве своего входа выходы нейронов предыдущего слоя – или, другими словами, признаки более высокого уровня формируются на основе признаков более низкого уровня.

Распространению данного подхода и, в связи с этим, очередному расцвету нейронных сетей, послужило три взаимосвязанных причины:

* Появление новых архитектур нейронных сетей, заточенных для решения определенных задач (сверточные сети, машины Больцмана и т.д.)
* Развитие и доступность вычислений с использованием gpu и параллельных вычислений в целом
* Появление и распространение подхода послойного обучения нейронных сетей, при котором каждый слой обучается отдельно с помощью стандартного алгоритма backpropagation (обычно на неразмеченных данных, то есть по сути происходит обучение автоэнкодера), что позволяет выявить существенные признаки на данном уровне, а затем все слои объединяются в единую сеть и происходит дообучение сети уже с применением размеченных данных для решения конкретной задачи (fine-tuning). Данный подход имеет два существенных преимущества. Во-первых, таким образом существенно повышается эффективность обучения сети, так как в каждый момент времени обучается не глубокая структура, а сеть с одним скрытым слоем – в результате исчезают проблемы с уменьшением значений ошибки по мере повышения глубины сети и соответствующим снижением скорости обучения. И во-вторых, данный подход к обучению сети позволяет использовать при обучении неразмеченные данные, которых обычно гораздо больше чем размеченных – что делает обучение сети более простым и доступным для исследователей. Размеченные данные в таком подходе требуются только в самом конце для донастройки сети на решение определенной задачи классификации, и при этом, поскольку общая структура признаков, описывающих данные, уже создана в процессе предыдущего обучения, для донастройки сети требуется значительно меньше данных, чем для первоначального обучения с целью выявления признаков. Кроме сокращения требуемого количества размеченных данных, использование подобного подхода позволяет обучить один раз сеть с использованием большого количества неразмеченных данных и затем использовать полученную структуру признаков для решения различных задач классификации, донастраивания сеть с помощью различных наборов данных – за гораздо меньшее время, чем потребовалось бы в случае полного обучения сети каждый раз.

# **1.5 Распознавание изображений**

Распознавание объектов на изображении является важным и интересным разделом компьютерной науки, имеющим самые различные применения в реальной жизни. Так, распознавание изображений может применяться для задачи распознавания автомобильных номеров, идентификации наличия пешеходов на дороге, поиска и распознавания рук на фотографиях, системах для управления каким-либо устройством жестами, проведения автоматического контроля качества на производстве – и это только очень небольшое подмножество практических применений распознавания изображений. Поэтому неудивительно, что интерес исследователей в области computer science и искусственного интеллекта к распознаванию изображений появился еще в 60-х годах 20 века и эта область науки активно развивается до сих пор.

Существует множество формулировок для задачи распознавания изображений, и определить ее однозначно достаточно сложно. Например, можно рассматривать распознавание изображений как задачу поиска и идентификации на исходном изображении некоторых логических объектов.

Распознавание изображений обычно является сложной задачей для компьютерного алгоритма. Это связано, в первую очередь, с высокой вариативностью изображений отдельных объектов. Так, задача поиска автомобиля на изображении является простой для человеческого мозга, который способен автоматически идентифицировать у объекта наличие важных для автомобиля признаков (колеса, специфическая форма) и при необходимости «достаивать» картинку в воображении, представляя недостающие детали, и крайне сложной для компьютера, так как существует огромное количество разновидностей автомобилей разных марок и моделей, имеющих во многом разную форму, кроме того, итоговая форма объекта на изображении сильно зависит от точки съемки, угла, под которым он снимается и других параметров. Также немаловажную роль играет освещение, которое оказывает влияние на цвет полученного изображения, а также может делать незаметными или искажать отдельные детали.

Таким образом, основные трудности при распознавании изображений вызывают:

* Вариативность предметов внутри класса
* Вариативность формы, размера, ориентации, положения на изображении
* Вариативность освещения

Для борьбы с этими трудностями на протяжении истории развития распознавания изображений были предложены самые различные методы, и в настоящее время в этой сфере уже удалось добиться существенного прогресса.

Первые исследования в области распознавания объектов на изображении были опубликованы в 1963 году Л.Робертсом в статье «Machine Perception Of Three-Dimensional Solids» [14], где автор сделал попытку абстрагироваться от возможных изменений в форме предмета и сконцентрировался на распознавании изображений простых геометрических форм в условиях различного освещения и при наличии поворотов. Разработанная им компьютерная программа была способна идентифицировать на изображении геометрические объекты некоторых простых форм и формировать их трехмерную модель на компьютере.

В 1987 году Ш.Улманом и Д.Хуттенлохером была опубликована статья «Object Recongnition Using Alignment» [15] где они также сделали попытку распознавания объектов относительно простых форм, при этом процесс распознавания был организован в два этапа: сначала поиск области на изображении, где находится целевой объект, и определение его возможных размеров и ориентации (“alignment”) с помощью небольшого набора характерных признаков, и затем попиксельное сравнение потенциального изображения объекта с ожидаемым.

Однако попиксельное сравнение изображений имеет множество существенных недостатков, таких как его трудоемкость, необходимость наличия шаблона для каждого из объектов возможных классов, а также то, что в случае попиксельного сравнения может осуществляться только поиск конкретного объекта, а не целого класса объектов. В некоторых ситуациях это применимо, однако в большинстве случаев все же требуется поиск не одного конкретного объекта, а множества объектов какого-либо класса.

Одним из важных направлений в дальнейшем развитии распознавания изображений стало идентификации контуров объекта на картинке. Во многих случаях именно контуры содержат большую часть информации об изображении, и в то же время рассмотрение изображения в виде совокупности контуров позволяет его существенно упростить.

Существует такой метод распознавания образов на изображении, как задача сегментации. Основная цель алгоритма – это выделение на изображении отдельных объектов, каждый из которых затем может быть персонально изучен и проклассифицирован. Задача сегментации значительно упрощается, если исходное изображение является бинарным – то есть состоит из пикселей только двух цветов. В этом случае задача сегментации часто решается с применением методов математической морфологии [16]. Суть методов математической морфологии заключается в представлении изображения как некоторого множества двоичных значений и применении к этому множеству логических операций, основные среди которых это перенос, наращивание и эрозия (рисунок 7). С применением данных операций и их производных, таких как замыкание и размыкание, появляется возможность, например, устранить шум на изображении или выделить границы. Если подобные методы применяются в задаче сегментации, то наиболее важной их задачей становится как раз-таки задача устранения шума и формирования на изображении более-менее однородных участков, которые затем легко найти с помощью алгоритмов, аналогичных поиску связных компонент в графе – это и будут искомые сегменты изображения.

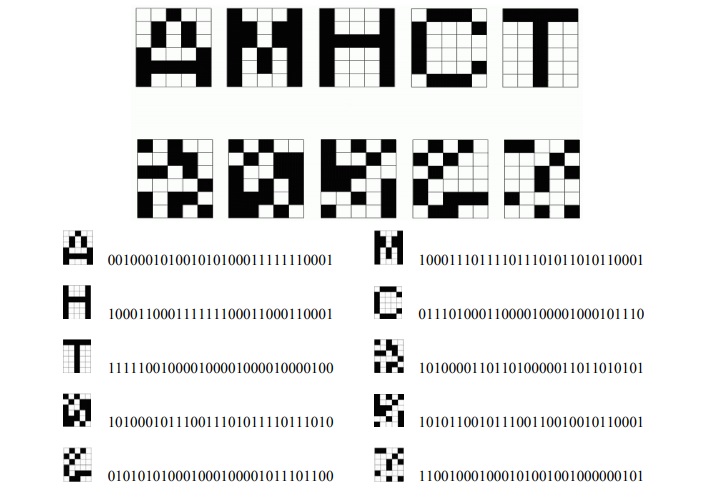


Рисунок 7

Что касается сегментации RGB-изображений, то одним из важных источников информации о сегментах изображения может стать его текстура. Для определения текстуры изображения часто применяется фильтр Габора [17], который был создан в попытках воспроизвести особенности восприятия текстур человеческим зрением. В основе работы данного фильтра лежит функция частотного преобразования изображения.

Другое важное семейство алгоритмов, используемых для распознавания изображений – это алгоритмы, основанные на поиске локальных особенностей. Локальные особенности представляют собой некоторые хорошо различимые области изображения, которые позволяют соотнести изображение с моделью (искомым объектом) и определить, соответствует ли данное изображение модели и, если соответствует, определить параметры модели (например, угол наклона, примененное сжатие и т.д.). Для качественного выполнения своих функций локальные особенности должны быть устойчивы к афинным преобразованиям, сдвигам и т.д. Классическим примером локальных особенностей являются углы, которые часто присутствуют на границах различных объектов. Наиболее популярным алгоритмом для поиска углов является детектор Харриса [18].

В последнее время все большей популярностью пользуются методы распознавания изображений, основанные на нейронных сетях и глубоком обучении. Основной расцвет этих методов наступил после появления в конце 20 века сверточных сетей LeCun, которые показывают значительно лучшие результаты в распознавании изображений по сравнения с остальными методами. Так, большая часть лидирующих (и не только) алгоритмов в ежегодном соревновании по распознаванию изображений ImageNet-2014 использовала в том или ином виде сверточные сети.

# **1.5.1 Каскады Хаара**

Одним из наиболее популярных и используемых признаков для распознавания образов являются каскады Хаара [19]. Каждый признак включает в себя несколько смежных прямоугольных зон, расположенных определенным образом на изображении. Разница между суммами яркостей пикселей зон, является значением признака. Различные признаки имеют разное расположение более светлых и более темных зон, таким образом позволяя описать одним из них практически любое сочетание данных зон.

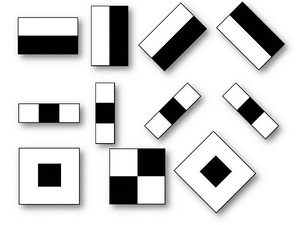


Рисунок 8. Каскады Хаара

Алгоритм обнаружения объектов с применением признаков Хаара был впервые описан Паулом Виолой и Майклом Джонсом в 2001 году [20].

Каскады Хаара - это набор из некоторого количества примитивов Хаара, имеющий определенные свойства взаимного расположения признаков на изображении. Используя достаточно большое количество примитивов, можно описать любой объект на изображении.

Эта техника удобна тем, что позволяет использовать предварительно рассчитанное описание характеристик искомого изображения многократно и является одним из самых быстрых способов распознавания объектов на изображении при условии использования предварительно рассчитанного каскада.

Принцип работы алгоритма заключается в том, что по всему изображению прогоняется прямоугольное сканирующее окно. Во всех тестируемых положениях рассчитывается яркость зон, описанных в примитивах поискового каскада. Если карта яркости фрагмента изображения в окне похожа на искомый примитив - информация об этом сохраняется. По итогам сканирования изображения найденные совпадения анализируются, группируются и проверяются на соответствие условиям взаимного расположения элементов в каскаде. Если расположение найденных элементов совпадает с эталонным, зоны, содержащие требуемые признаки, подаются на выход алгоритма.

Помимо высокой скорости распознавания преимуществом использования данного алгоритма является то, что каскады для большинства основных фигур и предметов, включая части человеческого тела, уже неоднократно были рассчитаны с достаточной точностью, что позволяет не проводить обучение и подбор соответствующих выборок самостоятельно, а просто взять готовый классификатор и использовать в своей программе.

# **2. ИСПОЛЬЗУЕМЫЕ СРЕДСТВА ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ**

# **2.1 Язык программирования Python**

# **2.1.1 История языка**

Разработка началась в 1980-х году, а завершилась она в 1991. Язык Python [21] был создан Гвидо ван Россумом. Хоть основным символом «Питона» является змея, название он получил в честь комедийного американского шоу. При создании языка разработчик использовал некоторые команды, заимствованные уже у существующих Pascal, С и С++. После выхода в интернет первой официальной версии целая группа программистов присоединилась к его доработке и улучшению. Одним из факторов, которые позволили стать «Питону» достаточно известным, является дизайн. Многими весьма успешными специалистами он признается одним из лучших. - Читайте подробнее на FB.ru:

# **2.1.2 Синтаксис языка**

Язык программирования Python для начинающих специалистов станет отличным учителем. Он имеет достаточно простой синтаксис.

* Конец строки является концом инструкции (точка с запятой не требуется);
* Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков. Используется знак табуляции для отступа
* Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции;
* Однострочные комментарии начинаются со знака фунта «#», многострочные — начинаются и заканчиваются тремя двойными кавычками «"""»;
* Огромное количество модулей, как относящихся к стандартным настройкам Python, так и синхронизированные сторонние.

Именно этот язык программирования мы будем использовать в качестве написания программы для идентификации руки на изображении. Для более удобной и продуктивной работы нам понадобиться скачать IDE - (Integrated Development Environment) – интегрированная среда разработки.

# **2.2 Интегрированная среда разработки PyCharm**

Существует достаточно много IDE сред для разных языков программирования. Допустим, самой известной на сегодняшний день средой разработки на С++ является Visual Studio. Многие используют более простые и не менее полезные Eclipse или NetBeans.

PyCharm – это самая интеллектуальная Python IDE с полным набором средств для эффективной разработки на языке Python [22]. Выпускается в двух вариантах – бесплатная версия PyCharm Community Edition и поддерживающая больший набор возможностей PyCharm Professional Edition. PyCharm быстро выполняет инспекцию кода, автодополнение, в том числе основываясь на информации, полученной во время исполнения кода, навигацию по коду, обеспечивает множество рефакторингов.

Ключевые возможности программы

* Мощный и функциональный редактор кода с подсветкой синтаксиса, авто-форматированием и авто-отступами для поддерживаемых языков;
* Простая и мощная навигация в коде;
* Помощь при написании кода, включающая в себя автодополнение, авто-импорт, шаблоны кода, проверка на совместимость версии интерпретатора языка, и многое другое. В качестве примера взята библиотека numpy; (Рисунок 9)

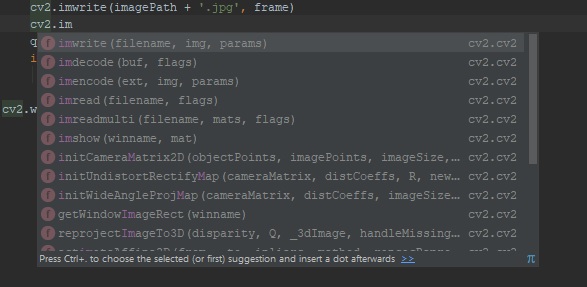


Рисунок 9

* Быстрый просмотр документации для любого элемента прямо в окне редактора, просмотр внешней документации через браузер. Открыть документацию в редакторе можно с помощью Ctrl+Q. Далее нажать на любой элемент в программе и в окне справа появится документация об интересующей команде; (Рисунок 10)

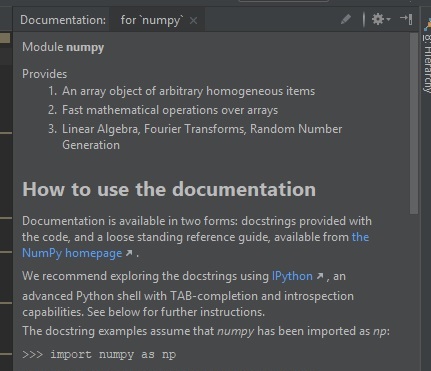


Рисунок 10

* Большое количество инспекций кода;
* Интеграция с системами контроля версий (VCS). Находится в верхней панели навигации редактора;
* Поддержка схем наиболее популярных IDE/редакторов. таких как Netbeans, Eclipse, Emacs, эмуляция VIM редактора;
* Поддерживаемые языки: Python (Versions: 2.x, 3.x), Jython, Cython, IronPython, PyPy, Javascript, CoffeScript, HTML/CSS, Django/Jinja2 templates, Gql, LESS/SASS/SCSS/HAML, Mako, Puppet, RegExp, Rest, SQL, XML, YAML;
* PyCharm имеет несколько цветовых схем, а также настраиваемую подсветку синтаксиса кода. Для выбора цветовой схемы заходим File > Settings > Editor > Color Scheme; (Рисунок 11)

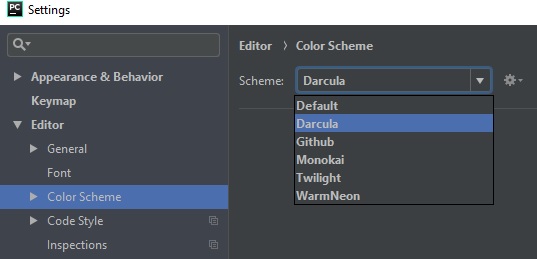


Рисунок 11

* Огромная, постоянно пополняемая коллекция плагинов;
* Кросс-платформенность (Windows, Mac OS X, Linux).

**2.3 Библиотека OpenCV**

OpenCV - самая популярная библиотека компьютерного зрения. Она написана на C/C++, ее исходный код открыт [23]. Библиотека включает более 1000 функций и алгоритмов. Она разрабатывается c 1998 г., сначала в компании Интел, теперь в Itseez при активном участии сообщества. О высокой популярности библиотеки свидетельствует количество загрузок, их более 6000000 загрузок (без учета svn/git трафика).

Библиотека распространяется по лицензии BSD, что означает ее свободное и бесплатное использовать как в открытых проектах с открытым кодом, так и в закрытых, коммерческих проектах. Библиотеку не обязательно копировать целиком в свой проект, можно использовать куски кода. Единственное требование лицензии – наличие в сопровождающих материалах копии лицензии OpenCV.

Из-за либеральной лицензии библиотека используется многими компаниями, организациями, университетами, например, NVidia, Willow Garage, Intel, Google, Stanford University. Компании NVidia и WillowGarage частично спонсируют ее разработку.

Библиотека состоит из 16 модулей. Реализованы около 1000 алгоритмов. Функциональность доступна на разных языках: C, C++, Python, CUDA, Java. Поддерживаются основные операционные системы: MS Windows, Linux, Mac, Android, iOS. Есть возможность использования сторонних библиотек.

Обратим внимание на поддержку работы с xml-файлами. Результаты промежуточных вычислений можно сохранять в xml-файлах, а затем их прочитать, например, в другой программе, что облегчает разработку алгоритма по частям, работу над алгоритмом целой командой.

Заметим, что библиотека OpenCV реализует, как правило, только базовые операции, используемые в компьютерном зрении. Таким образом, ее можно рассматривать как в целом низкоуровневую библиотеку компьютерного зрения. Для решения серьезных задач необходимо на основе предоставленных библиотекой кирпичиков создавать свои сложные приложения.

# **2.3.1 OpenCV и документация по библиотеке**

На сайте библиотеки находится много полезный данных по использованию программы [24]. На нем можно загрузить библиотеку для разных платформ, краткие инструкции по установке (Quick Start), уроки по ее использованию (Tutorials), справочную карту (Cheat Sheet), которую было бы неплохо сразу распечатать, ссылки на различные сайты и тому подобное. Ссылка WIKI ведет на репозиторий в Github.

На сайте собрана вся документация по работе с библиотекой. В частности, имеются подробные описания всех функций с полнотекстовым поиском.

Также есть возможность задать вопрос, касающийся библиотеки, и получить на него ответ от разработчиков или других пользователей. Прежде чем сформулировать свой вопрос, имеет смысл попытаться найти похожий, среди тех, которые уже задавали. Наиболее популярные вопросы попадают в верхнюю часть списка. На них нужно обратить внимание в первую очередь.

# **2.3.2 OpenCV на Python**

Для первых экспериментов с библиотекой авторы рекомендуют использовать Python. Программу на этом языке можно писать в любом текстовом редакторе. Далее без предварительной компиляции она выполняется интерпретатором Python. С поддержкой библиотеки NumPy Python превращается в бесплатный аналог системы MATLAB, при этом сам язык более продвинутый, чем язык MATLAB.

# **3. ПРОЦЕСС ИССЛЕДОВАНИЯ**

Для реализации задачи распознавания, а именно детектирования кисти руки на изображении при помощи нейронной сети, потребуется выполнить следующие действия:

* Выбор языка программирования
* Установка библиотек
* Поиск исходных данных
* Обработка исходных данных
* Запуск
* Анализ результатов

В ходе тщательного изучения всех составляющих и особенностей языков программирования принято решение написания программы на Python. Такой выбор можно аргументировать тем, что нейронные сети на Python более просты для восприятия, нежели на других языках.

Для облегчения работы установим PyCharm и загрузим туда несколько важных модулей:

* Pip – Система управления пакетами
* OpenCV – библиотека компьютерного зрения
* Numpy – модуль, для добавления больших многомерных массивов и матриц, и огромного количества математических функций для операций с массивами

В качестве входных данных для исследования будем использовать картинки. Так как найти такое количество фотографий с изображением кистей рук сжатой в кулак не представляется возможным принято решение провести сбор данных вручную. Удалось собрать 300 изображений с разными размерами и количеством пикселей. Визуально выборка выглядит следующим образом (рис. 12, 13, 14):



Рисунок 12 Рисунок 13 Рисунок 14

Перед началом работы исходные данные были соответствующим образом подготовлены и преобразованы в формат jpg. Для удобства все изображения помещены в одну папку.

Первым делом реализуем задачу распознавания объекта с помощью библиотеки OpenCV. Будем использовать скрипт, который относится к формату xml, для обнаружения руки на изображении на основе признаков Хаара (см. приложение 1, 2). Получим на выходе изображение с выделением нужного нам объекта (рис. 15, 16, 17).

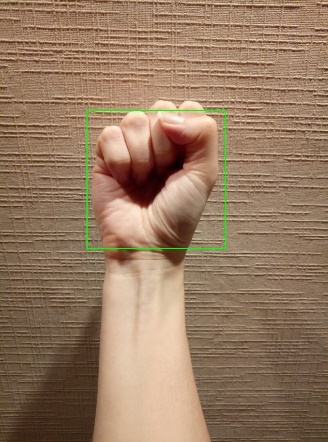
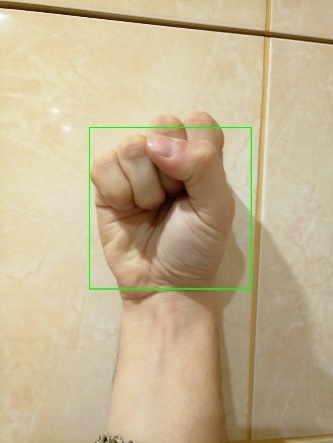
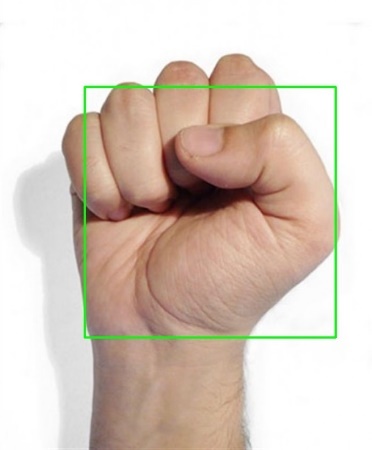


Рисунок 15 Рисунок 16 Рисунок 17

Необходимо узнать координаты выделенной области на изображении и записать в data.csv для сравнения с поиском объекта при помощи OpenCV и нейронной сетью. Программа сканирует изображение с верхнего левого угла до правого нижнего. Поэтому для понимания укажем:

* x – расстояние от левого края изображения в пикселях
* y – расстояние от верхнего края
* w и h – толщина и высота зеленого квадрата

Координаты некоторых изображений предоставлены в таблице №1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File | x | y | w | h |
| data\1.jpg | 218 | 310 | 394 | 394 |
| data\2.jpg | 117 | 197 | 180 | 180 |
| data\3.jpg | 295 | 380 | 325 | 325 |
| data\4.jpg | 104 | 54 | 137 | 137 |
| data\5.jpg | 269 | 166 | 128 | 128 |
| data\6.jpg | 121 | 133 | 196 | 196 |
| data\7.jpg | 125 | 50 | 120 | 120 |
| data\8.jpg | 50 | 102 | 301 | 301 |
| data\9.jpg | 153 | 84 | 63 | 63 |
| data\10.jpg | 127 | 70 | 60 | 60 |

Таблица №1

Создадим два zip архива. Первый с выборкой из изображений без выделенного объекта, а второй с обнаруженным объектом.

Стоит отметить, что для решения данной задачи были выбраны сверточные нейронные сети, подробно описанные в теоретической части работы.

Все исходные данные для запуска программы подготовлены. Далее необходимо подключить несколько библиотек для работы с нейронной сетью.

* SciPy – создана для инженерных расчетов.
* Pillow – предназначена для работы с изображениями
* Pandas – работа с данными
* Matplotlib – для визуализации данных двумерной графикой
* Scikit-learn – библиотека предназначена для моделирования данных. Огромным набор команд для настройки нейронной сети с помощью алгоритма обучения с учителем
* Chainer – китайская программная платформа, для работы с нейронными сетями. Для вычисления использует архитектуру Cuda.

Запускаем обучение нейронной сети (Приложение 4), а после этого главную программу (Приложение 5).

На выходе получим изображения размером 48 на 48 с выделенными объектами при помощи нейронной сети и OpenCV (рис. 18, 19, 20). Также отметим, что 80% изображений идет на обучение и только 20 подается на выход.

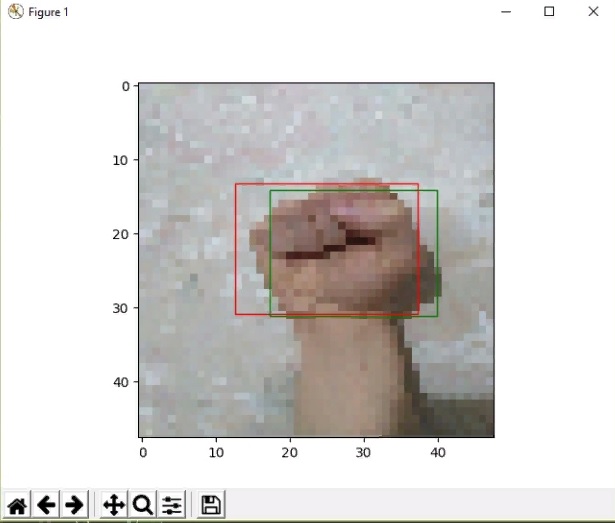


Рисунок 18

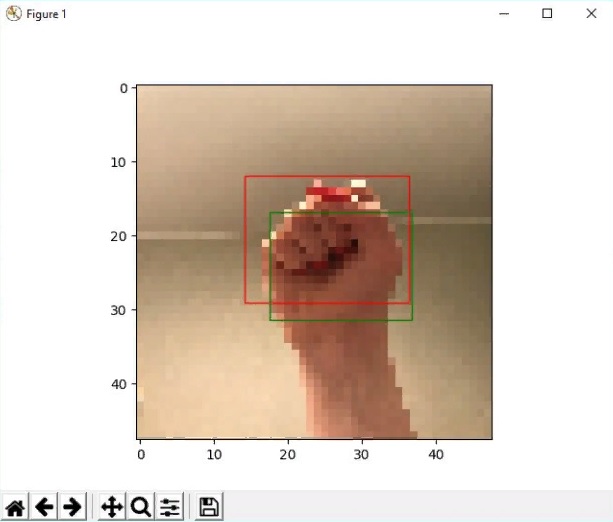


Рисунок 19

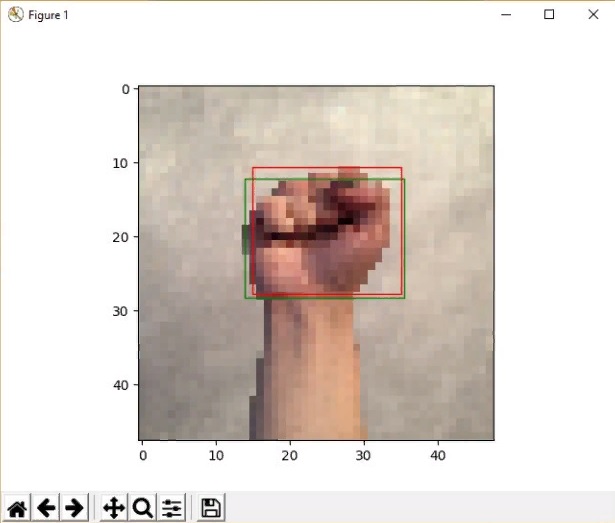


Рисунок 20

Подводя итоги, можно сказать, что с помощью сверточных нейронных сетей, можно получить неплохие результаты в обнаружении кисти руки на изображении. Также хотелось бы отметить достаточно высокое качество выполнения задачи нахождения объекта при помощи библиотеки компьютерного зрения OpenCV и использования в ней каскадов Хаара.

# **ЗАКЛЮЧЕНИЕ**

В данной работе была подробно изучена задача распознавания объекта на изображении с применением искусственных нейронных сетей. Были рассмотрены наиболее актуальные в настоящее время подходы к распознаванию изображений, в том числе использование глубоких нейронных сетей, а также написана программа для распознавания изображений на примере задачи обнаружения кисти руки сжатой в кулак с применением глубоких сетей. По результатам работы можно сказать, что все поставленные в начале работы задачи были выполнены:

1. Был проведен аналитический обзор литературы по теме применения искусственных нейронных сетей для распознавания изображений. По результатам данного обзора было выяснено, что наиболее эффективными и распространенными в последнее время являются подходы к распознаванию изображений, основанные на применении глубоких сверточных сетей
2. Была проанализированы различные методы распознавания объекта на изображении. В ходе освоения материала на практике, впечатляющий результат показала сверточная нейронная сеть, а также библиотека OpenCV с добавлением каскадов Хаара.
3. Была разработана программа, которая обнаруживает кисть руки на изображении с достаточно высокой точностью.

В целом, данная работа подтвердила, что в настоящее время глубокие искусственные нейронные сети, в особенности сверточные сети, являются наиболее результативным и перспективным подходом для детектирования изображений, что подтверждается результатами многочисленных исследований и проводимых соревнований по распознаванию изображений.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Abdelhamid M. Machine Learning / Abdelhamid Mellouk, Abdennacer Chebira. - InTech. 2009. – 450 с.
2. Mitchell T. Generative and discriminative classifiers: naive Bayes and logistic regression, - М.: Иностр. лит., 2015. 17 – c.
3. Мак-Каллок У. Логическое исчисление идей, относящихся к нервной деятельности. / Уоррен Мак-Каллок, Уолтер Питтс // The bulletin of mathematical biophysics. - 1954. №4. – С.115-133.
4. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain / Frank Rosenblatt // Psychological review. – 1958. – № 6. – С. 386.
5. Минский, М., Персептроны = Perceptrons / Минский, М., Пейперт, С. — М.: Мир, 1971. — 261 с.
6. Галушкин, А. И. Синтез многослойных систем распознавания образов / А. И. Галушкин. – М.: Энергия, 1974. – 368 с.
7. Werbos P. J., Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.
8. LeCun Y. Convolutional neural networks for Images. – <http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf>
9. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position / Kunihiko Fukushima // Biological Cybernetics. – 1980. - №36[4]. - С. 193-202.
10. Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition / Kunihiko Fukushima // Neural Networks. – 1988. №1[2] - C. 119-130.
11. Fukushima K. Neocognitron for handwritten digit recognition / Kunihiko Fukushima // Neurocomputing. – 2003. №51, - C. 161-180.
12. LeCun Y. Large-scale learning with SVM and Convolutional for Generic Object Categorization / Y. LeCun, F.J. Huang // Computer Society Conference on Computer Vision and Pattern Recognition. – 2006. №1. – С. 284-291.
13. Goodfellow I. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. - The MIT Press. 2016. – 800 с.
14. Roberts L. Machine Perception Of Three-Dimensional Solids / Roberts L. Thesis (Ph. D.) -- Massachusetts Institute of Technology, Dept. of Electrical Engineering. -1963. – 81 c.
15. Huttenlocher D. P. Object recognition using alignment / Huttenlocher D. P., Ullman S. // Proc. ICCV. – 1987. – №87. – С. 102-111.
16. Serra J. Introduction to mathematical morphology / Serra J. //Computer vision, graphics, and image processing. – 1986. – №. 3. – С. 283-305.
17. Сойфер В.А. Методы компьютерной обработки изображений / Сойфер В.А // Физматлит, - 2003. — С. 459.
18. Tracking and Recognation of Objects using SURF Descriptor and Harris corner Detection. – <http://inpressco.com/wp-content/uploads/2014/04/Paper58775-778.pdf>
19. Messom C.H. Fast and Efficient Rotated Haar-like Features Using Rotated Integral Images / Messom, C.H., Barczak, A.L.C // Australian Conference on Robotics and Automation (ACRA2006). – 2006. C. 1-6.
20. Rapid object detection using a boosted cascade of simple features. - <http://www.merl.com/publications/docs/TR2004-043.pdf>
21. Маккинли У. Python и анализ данных / Маккинли У. — Перевод с английского. — М.: ДМК Пресс, 2015. — 482 с.
22. Интегрированная среда разработки Pycharm. - <http://www.jetbrains.com/pycharm/>
23. Кэлер А. Изучаем OpenCV 3 = Learning OpenCV 3 / Кэлер А., Брэдски Г. — М.: ДМК-Пресс, 2017. — 826 с.
24. Open Computer Vision library. - <http://opencv.org/>

# **ПРИЛОЖЕНИЕ 1**

import cv2

import numpy as np

hand\_cascade = cv2.CascadeClassifier('Путь\_к\_файлу\_скрипта, формата xml')

frame = cv2.imread("Путь\_к\_изображению")

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

hands = hand\_cascade.detectMultiScale(gray, 1.1 , 5)

for(x, y, w, h) in hands:

cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('hands', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# **ПРИЛОЖЕНИЕ 2**

Небольшой отрывок скрипта

<?xml version="1.0"?>

<opencv\_storage>

<A\_gest type\_id="opencv-haar-classifier">

<size>

24 24</size>

<stages>

<\_>

<!-- stage 0 -->

<trees>

<\_>

<!-- tree 0 -->

<\_>

<!-- root node -->

<feature>

<rects>

<\_>

3 3 9 16 -1.</\_>

<\_>

3 7 9 8 2.</\_></rects>

<tilted>0</tilted></feature>

<threshold>-0.0223442204296589</threshold>

<left\_val>0.7737345099449158</left\_val>

<right\_val>-0.9436557292938232</right\_val></\_></\_>

<\_>

<!-- tree 1 -->

<\_>

<!-- root node -->

<feature>

<rects>

<\_>

0 9 12 5 -1.</\_>

<\_>

6 9 6 5 2.</\_></rects>

<tilted>0</tilted></feature>

<threshold>-9.3714958056807518e-003</threshold>

<left\_val>0.5525149106979370</left\_val>

<right\_val>-0.9004204869270325</right\_val></\_></\_></trees>

<stage\_threshold>-0.3911409080028534</stage\_threshold>

<parent>-1</parent>

<next>-1</next></\_>

<\_>

# **ПРИЛОЖЕНИЕ 3**

import cv2

import numpy as np

imagePath = ('Путь к изображению')

hand\_cascade = cv2.CascadeClassifier('Путь к файлу скрипта, формата xml')

frame = cv2.imread(imagePath)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

hands = hand\_cascade.detectMultiScale(gray, 1.1 , 5)

for(x, y, w, h) in hands:

print(imagePath, x, y, w, h)

cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('hands', frame)

cv2.imwrite(imagePath + '.jpg', frame)

q = cv2.waitKey(33)

if q == 27:

break

cv2.waitKey(0)

# **ПРИЛОЖЕНИЕ 4**

Листинг обучения

import numpy as np  
import chainer  
from chainer.backends import cuda  
from chainer import Function, gradient\_check, report, training, utils, Variable  
from chainer import datasets, iterators, optimizers, serializers  
from chainer import Link, Chain, ChainList  
import chainer.functions as F  
import chainer.links as L  
  
  
from chainer.training import extensions  
def augment(x, y):  
 angles = np.random.choice([0, 1, 2, 3], len(x))  
 for i in range(1,4):  
 index = np.where(angles == i)[0]  
 x[index] = np.rot90(x[index], i, (2, 3))  
 if i%2 != 0:  
 c = y[index, 3]  
 y[index, 3] = y[index, 2]  
 y[index, 2] = c  
 theta = np.radians(90 \* i)  
 c, s = np.cos(theta), np.sin(theta)  
 R = np.array(((c, -s), (s, c)))  
 y[index, :2] = np.dot(y[index, :2], R)  
  
  
class FaceNet():  
 def \_\_init\_\_(self):  
 self.model = Chain(  
 conv1=L.Convolution2D(3, 20, 3, 1, 1),  
 conv2=L.Convolution2D(20, 20, 3, 1, 1),  
  
 conv3=L.Convolution2D(20, 40, 3, 1, 1),  
 conv4=L.Convolution2D(40, 40, 3, 1, 1),  
  
 linear1=L.Linear(None, 100),  
 linear2=L.Linear(100, 4)  
 )  
  
 self.optimizer = optimizers.Adam()  
 self.optimizer.setup(self.model)  
  
 def foward(self, x):  
 out = self.model.conv1(x)  
 out = F.elu(out)  
 out = self.model.conv2(out)  
  
 out = F.max\_pooling\_2d(out, 2)  
 out = F.elu(out)  
 out = self.model.conv3(out)  
 out = F.elu(out)  
 out = self.model.conv4(out)  
 out = F.elu(out)  
  
 out = F.average\_pooling\_2d(out, 6)  
 out = F.dropout(out)  
 out = self.model.linear1(out)  
 out = F.elu(out)  
 out = F.dropout(out)  
 out = self.model.linear2(out)  
  
  
 return out  
  
 def predict(self, X, step=100):  
 with chainer.using\_config('train', False):  
 with chainer.no\_backprop\_mode():  
 output = []  
 for i in range(0, len(X), step):  
 x = Variable(X[i:i + step])  
 output.append(self.foward(x).data)  
 return np.vstack(output)  
  
 def score(self, X, Y, step=100):  
 predicted = self.predict(X, step)  
 score = F.r2\_score(predicted, Y).data  
 return score  
  
  
  
 def fit(self, X, Y, batchsize=100, n\_epoch=10):  
 with chainer.using\_config('train', True):  
 learning\_curve = []  
 for epoch in range(n\_epoch):  
 print('epoch ',epoch)  
 index = np.random.permutation(len(X))  
 for i in range(0, len(index), batchsize):  
 self.model.cleargrads()  
 print(i)  
 x = X[index[i:i+batchsize]]  
 y = Y[index[i:i+batchsize]]  
 #augment(x, y)  
  
 x = Variable(x)  
 y = Variable(y)  
  
 output = self.foward(x)  
 loss = F.mean\_squared\_error(y, output)  
 loss.backward()  
  
 learning\_curve.append(float(loss.data))  
  
 self.optimizer.update()  
 return learning\_curve

# **ПРИЛОЖЕНИЕ 5**

Листинг программы

import numpy as np  
from matplotlib import pyplot as pl  
import pandas  
from sklearn.linear\_model import LinearRegression  
from neural\_network import FaceNet  
from PIL import Image  
import matplotlib.patches as patches  
data = pandas.read\_csv('data.csv', delimiter=' ')  
pathes = data[['File']].as\_matrix()[:, 0]  
rects = data[['x', 'y', 'w', 'h']].as\_matrix()  
size = 48  
imgX = []  
rectY = []  
  
  
for file, rect in zip(pathes, rects):  
 img = Image.open(file)  
 x = rect[0] / img.size[0]  
 y = rect[1] / img.size[1]  
 w = rect[2] / img.size[0]  
 h = rect[3] / img.size[1]  
 img = (np.array(img.resize((size, size)))/128.0 - 1).astype(np.float32).T  
 r = np.array([x, y, w, h], np.float32)  
 imgX.append(img)  
 rectY.append(r)  
  
imgX = np.array(imgX)  
rectY = np.array(rectY)  
rectY = (2 \* rectY - 1).astype(np.float32)  
  
index = np.random.permutation(len(imgX))  
train\_index = index[:int(len(index)\*0.8)]  
test\_index = index[int(len(index)\*0.8):]  
  
  
  
model = FaceNet()  
print('start fit')  
learning\_curve = model.fit(imgX[train\_index], rectY[train\_index], 20, 10)  
print('end fit')  
print(model.score(imgX[train\_index], rectY[train\_index]))  
print(model.score(imgX[test\_index], rectY[test\_index]))  
  
predicted\_rect = model.predict(imgX[train\_index])  
  
pl.figure()  
pl.plot(learning\_curve)  
pl.show()  
  
rectY = 0.5 \* (rectY + 1)  
predicted\_rect = 0.5 \* (predicted\_rect + 1)  
  
  
  
for img, rect, rect\_true in zip(imgX[test\_index], predicted\_rect, rectY[test\_index]):  
 image = (img.T + 1)/2  
 fig, ax = pl.subplots(1)  
 ax.imshow(image)  
 rect\_patch\_predicted = patches.Rectangle((rect[0]\*size, rect[1]\*size), rect[2] \* size, rect[3] \* size,  
 linewidth=1, edgecolor='r', facecolor='none')  
 rect\_patch\_true = patches.Rectangle((rect\_true[0] \* size, rect\_true[1] \* size), rect\_true[2] \* size, rect\_true[3] \* size,  
 linewidth=1, edgecolor='g', facecolor='none')  
  
 ax.add\_patch(rect\_patch\_predicted)  
 ax.add\_patch(rect\_patch\_true)  
 pl.show()