**1** ОБЗОР ЛИТЕРАТУРЫ

**1.1** Обзор аналогов

Для того, чтобы обосновать идею создания программного средства колоризации в данном подразделе будет рассмотрено готовое ПО, позволяющее колоризировать изображения.

Первым рассмотренным программным обеспечением является AKVIS Coloriage. AKVIS Coloriage – программа для раскрашивания черно-белых фотографий и замены цвета на цветных изображениях. Основная задача – разделение областей и выбор цветов. Согласно заявлению разработчиков, достаточно несколькими штрихами обозначить объекты для перекрашивания и запустить расчет. Coloriage распознает границы и раскрасит изображение, накладывая цвета с учетом текстур, освещенности и теней.



Рисунок 1.1 – Интерфейс ПО AKVIS Colorage

Программным обеспечением Colorage можно пользоваться как в виде отдельного продукта, так и в качестве плагина к Adobe Photoshop. Несмотря на заявление разработчиков об автоматической колоризации изображений, программе необходимо управление пользователя, который должен как выделить области, так и назначить цвета этим областям.

Несмотря на этот недостаток, достоинствами данного ПО являются удобный интерфейс для выделения областей и хороший инструмент для выбора цвета. Программа Coloriage поддерживает форматы JPEG, TIFF, BMP, PNG. AKVIS Colorage является ПО, распространяемым с помощью лицензии. однако предоставляется возможность пробного периода [1].

Adobe Photoshop является самым распространённым способом колоризации изображений. Данный факт объясняется широким распространением данного продукта. Adobe Photoshop является многофункциональным графическим редактором, разработанным и распространяемым фирмой Adobe Systems. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известным продуктом фирмы Adobe.



Рисунок 1.2 – Интерфейс ПО Adobe Photoshop

Несмотря на то, что изначально программа была разработана как редактор изображений для полиграфии, в данное время она широко используется и в веб-дизайне.

Несмотря на распространение данное программное обеспечение не является самым эффективным способом колоризации. Причиной тому является тот факт, что приложение не предоставляет никаких способов автоматической колоризации, а лишь предоставляет широкий выбор инструментов для изменения цвета объектов [2].

В процессе поиска способа полностью автоматической колоризации изображений была найдено веб-приложение, демонстрирующее возможности искусственного интеллекта в решении задачи колоризации [3].

Веб-приложение является частью открытой платформы Algorithmia, где разработчики могут разместить какой-либо алгоритм, предоставив API для использования другими разработчиками. Конкретное приложение предоставляет минимальный интерфейс, где пользователь может загрузить изображение либо предоставить ссылку на него.

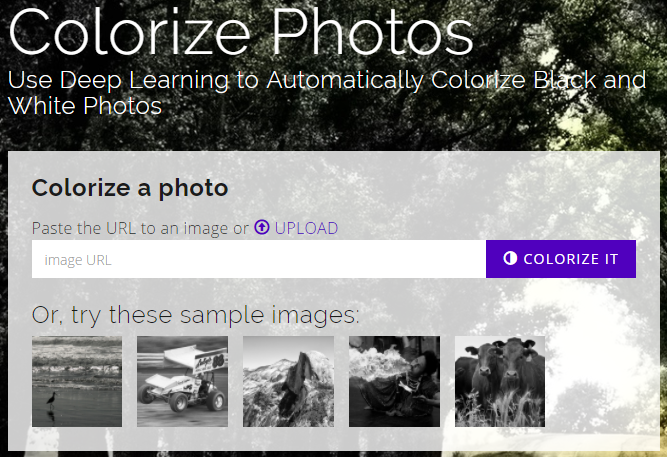


Рисунок 1.3 – Фрагмент веб-страницы с интерфейсом для колоризации

Также платформа предоставляет API для использования алгоритма разработчиками. Пример использования API на языке программирования Python:

import Algorithmia

input = {"image": "\_IMAGE\_URL\_"}

client = Algorithmia.client('\_API\_KEY\_')

algo = client.algo('deeplearning/ColorfulImageColorization/1.0.1')

# prints {"output": "data://.algo/temp/image.png"}

print algo.pipe(input)

Как видно из примера, для использования API необходим ключ, который можно приобрести, используя платформу Algorithmia.

Несмотря на тот факт, что данное ПО предоставляет способ полностью автоматической колоризации изображений, оно не является свободным, а минимальный интерфейс веб-приложение недостаточно удобен.

**1.2** Выбор архитектуры нейронной сети

На первом этапе разработки и проектирования системы была проведена работа по поиску подходящей для цели проекта архитектуры нейронной сети. Для этого был произведён поиск некоего подобия каталога архитектур нейронных сетей. Из найденного каталога, рассмотрим архитектуры сетей, которые спроектированы для обработки изображений. Одним из вариантов таких сетей является свёрточная сеть (рисунок 1.4).

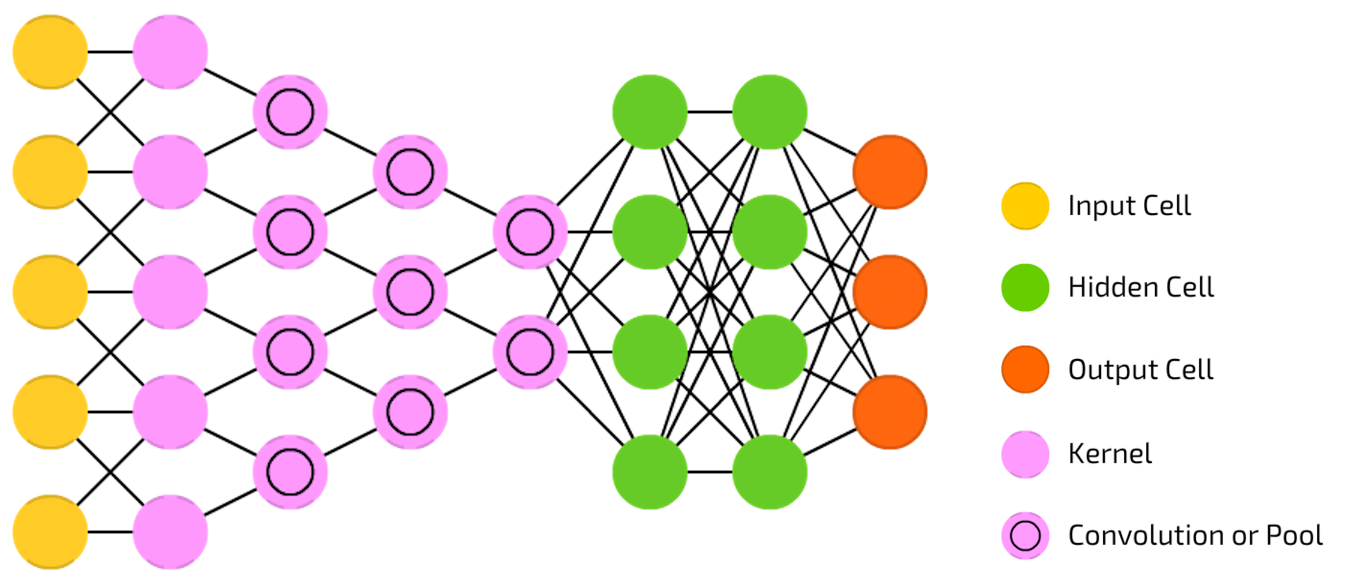


Рисунок 1.4 – Архитектура свёрточной нейронной сети

Свёрточные нейронные сети кардинально отличаются от других нейронных сетей. Они используются в основном для обработки изображений, иногда для аудио и других видов входных данных. Типичным способом применения сети является классификация изображений. Свёрточные нейронные сети используют операцию свёртки для фрагментов изображений, что даёт преимущество в производительности перед обычными сетями прямого распространения с полносвязными слоями [4]. Входные данные передаются через свёрточные слои, которые не являются полносвязными. Эти слои имеют свойство сжиматься с глубиной. Обычно они уменьшаются на делитель количества входных данных. Подобным делителем может быть степень двойки.

Кроме свёрточных слоев есть также так называемые слои объединения. Этот слой является хорошим способом уменьшить размерность получаемых данных, что в свою очередь опять же увеличивает производительность обучения и обработки сетью данных. После нескольких чередований слоёв свёртки и объединения, обработанные данные поступают на многослойный персептрон с относительно небольшим количеством скрытых слоёв сети. Такие сети называются глубокими свёрточными сетям, что обычно опускается на практике.

Другим вариантом глубокие свёрточные обратные глубинные сети (рисунок 1.5)

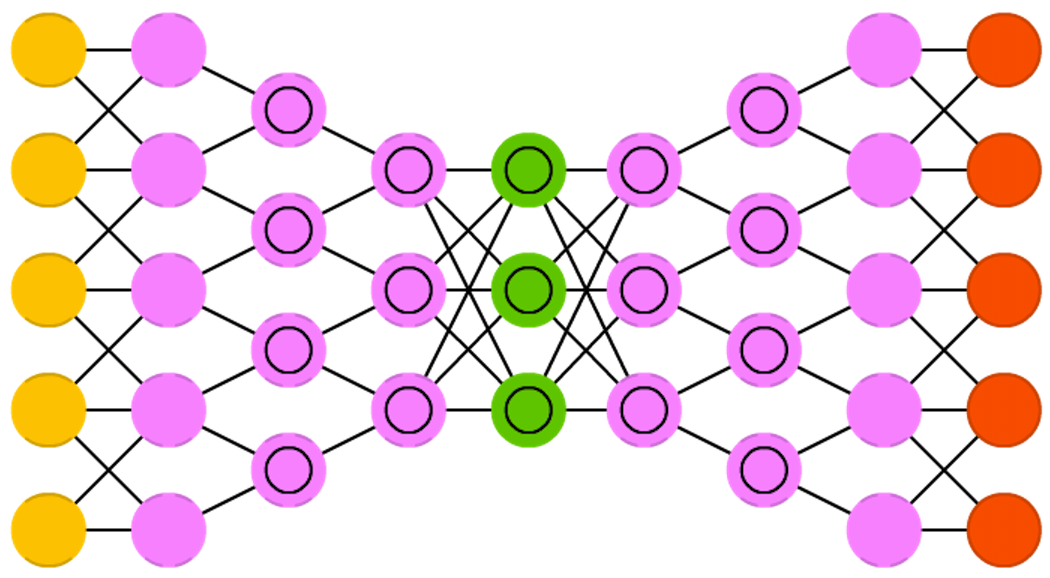


Рисунок 1.5 – Архитектура свёрточной обратной нейронной сети

Сеть напоминает вариационные автоэнкодеры со свёрточными и развертывающими сетями в качестве кодирующей и декодирующей частей соответственно. Такие сети представляют черты изображения в виде вероятностей и могут научиться строить, например, изображение кошки и собаки вместе, «взглянув» лишь на картинки только с кошками и только с собаками. Разработчики также продемонстрировали, что подобные сети могут моделировать различные сложные преобразования изображений, например, изменение источника света или поворот 3D объектов. Такие сети обычно обучают методом обратного распространения. Минусом подобной архитектуры, для решения данной задачи является тот факт, что на выходе нейронной сети само исходное изображение может достаточно сильно изменяться. Это связано с тем фактом, то в середине нейронной сети остаётся информация только о контексте изображения.

Третьим рассмотренным вариантом будет классический многослойный персептрон.

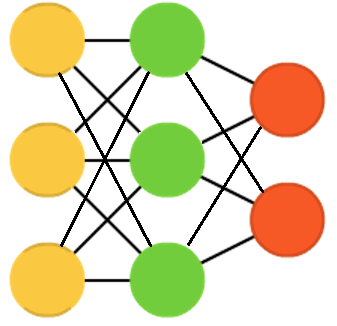


Рисунок 1.6 – Архитектура многослойного персептрона

Персептроны очень просты – они передают информацию от входа к выходу. Считается, что у нейронных сетей есть слои, каждый из которых состоит из входных, скрытых или выходных нейронов. Нейроны одного слоя между собой не связаны, при этом каждый нейрон этого слоя связан с каждым нейроном соседнего слоя. Простейшая мало-мальски рабочая сеть состоит из двух входных и одного выходного нейрона и может моделировать логический вентиль – базовый элемент цифровой схемы, выполняющий элементарную логическую операцию. FFNN обычно обучают методом обратного распространения ошибки, подавая модели на вход пары входных и ожидаемых выходных данных. Под ошибкой обычно понимаются различные степени отклонения выходных данных от исходных (например, среднеквадратичное отклонение или сумма модулей разностей). При условии, что сеть обладает достаточным количеством скрытых нейронов, теоретически она всегда сможет установить связь между входными и выходными данными. На практике использование сетей прямого распространения ограничено, и чаще они используются совместно с другими сетями. Данный факт связан с уменьшением производительности при увеличении количества нейронов и количества слоёв. Из-за того, что каждый нейрон предыдущего слоя связан с каждым нейроном следующего, количество связей настолько велико, что реализация достаточно большого многослойного персептрона требует огромных затрат производительности.

Из рассмотренных вариантов было принято решение использовать свёрточную нейронную сеть. Выбор обусловлен тем фактом, что ни один из аналогов не подходит для реализации сети для колоризации полутоновых изображений: многослойный персептрон имеет слишком много связей, каждая из которых имеет свой вес, который должен учитываться при прямом проходе через сеть; свёрточной обратной сети в свою очередь необходим длительный процесс обучения, для того, чтобы сеть выдавала подобные оригинальному изображения.

**1.3** Обзор вариантов обучения нейронной сети

Свёрточной нейронной сети, как и любой другой нейронной сети, необходимо обучение. Обучение нейронных сетей может происходить различными способами:

* обучение с учителем;
* обучение без учителя;
* смешанное обучение.

Наиболее типичным вариантом обучения для свёрточных нейронных сетей является обучение с учителем. Подобный вид обучения подразумевает наличие алгоритма, которые выполняет обучение нейронной сети. Для того, чтобы обучающий алгоритм мог выполнять свою работу необходима база подготовленных заранее изображений, которая являлась бы образцом того, как должна работать нейронная сеть. Поиск базы изображений был остановлен ImageNet.

ImageNet – это проект по созданию и сопровождению массивной базы данных аннотированных изображений, предназначенной для отработки и тестирования методов [распознавания образов](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2) и [машинного зрения](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%B7%D1%80%D0%B5%D0%BD%D0%B8%D0%B5). По состоянию на 2016 год в базу данных было записано около десяти миллионов URL с изображениями, которые прошли ручную аннотацию для ImageNet. В аннотациях перечислялись объекты, попавшие на изображение, и прямоугольники с их координатами. База данных с аннотацией и URL изображений от третьих лиц доступна непосредственно через ImageNet, но при этом сами изображения не принадлежат проекту [5]. Преимуществом данной базы изображений является наличие дерева изображений структурированного по контексту самого изображения (рисунок 1.7). Возможность получения изображений с определённым контекстом позволит в дальнейшем сделать нейронную сеть специализированной на обработке изображений с соответствующим контекстом. Такой подход к использованию программного средства позволит уменьшить время обучения сети и быстрее ввести продукт в эксплуатацию.

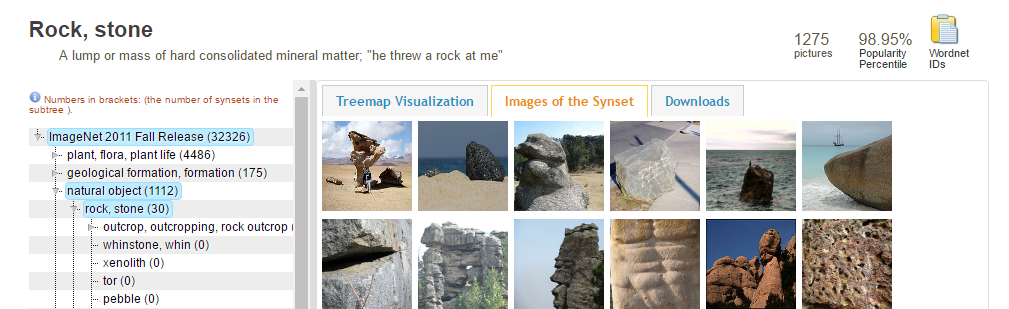


Рисунок 1.7 – Фрагмент веб-страницы базы изображений ImageNet, содержащий дерево изображений, структурированных по контексту

**1.4** Выбор средства построения нейронной сети

Построение нейронной сети является важным шагом в реализации проекта. Сам процесс построения нейронной сети должен быть простым и удобным, чтобы сохранить время, потраченное на создание сети и сконцентрироваться на других задачах. С этой целью был произведён поиск различных средств, которые позволили бы сократить процесс создания нейронной сети.

В статье [6] представлены возможности наиболее популярных программных средств глубокого обучения (см. таблицу 1.1). Кроме того, рассмотрены некоторые из фреймворков, представленных в таблице.

Таблица 1.1 – Возможности программных средств глубокого обучения

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Язык | Поддерживаемые операционные системы | Поддержка свёрточных сетей |
| [DeepLearnToolbox](https://github.com/rasmusbergpalm/DeepLearnToolbox) | Matlab | Windows, Linux | Есть |
| [Theano](http://deeplearning.net/software/theano/) | Python | Windows, Linux, Mac | Есть |
| [Pylearn2](http://deeplearning.net/software/pylearn2/) | Python | Linux, Vagrant | Есть |
| [Deepnet](https://github.com/nitishsrivastava/deepnet) | Python | Linux | Есть |
| [Deepmat](https://github.com/kyunghyuncho/deepmat) | Matlab |  | Есть |
| [Torch](http://torch.ch/) | Lua, C | Linux, OS X, iOS, Android | Есть |
| [Darch](http://cran.r-project.org/web/packages/darch/index.html) | R | Windows, Linux | Нет |
| [Caff](http://caffe.berkeleyvision.org/)e | C++, Python, Matlab | Linux, OS X | Есть |
| [nnForge](http://milakov.github.io/nnForge/) | С++ | Linux | Есть |
| [CXXNET](https://github.com/antinucleon/cxxnet) | С++ | Linux | Есть |
| [Cuda-convnet](https://code.google.com/p/cuda-convnet/) | С++ | Linux, Windows | Есть |
| [Cuda CNN](http://www.mathworks.com/matlabcentral/fileexchange/24291-cnn-convolutional-neural-network-class) | Matlab | Linux, Windows | Есть |

Автор статьи [6] для дальнейшего рассмотрения выбрал четыре библиотеки: [Theano](https://habrahabr.ru/company/intel/blog/254747/#TheanoLib) и [Pylearn2](https://habrahabr.ru/company/intel/blog/254747/#Pylearn2Lib) являются одними из самых зрелых и функционально полных библиотек, [Torch](https://habrahabr.ru/company/intel/blog/254747/#TorchLib) и [Caffe](https://habrahabr.ru/company/intel/blog/254747/#CaffeLib) – широко используются сообществом. Из данного списка только Theano, Pylearn2 и Caffe поддерживают язык программирования Python и операционные системы Linux. Выбор был остановлен на библиотеке Caffe.

Caffe реализована с использованием языка программирования C++, имеются обертки на Python и MATLAB. Официально поддерживаемые операционные системы – Linux и OS X. Для ускорения вычислений Caffe может быть запущена на GPU с использованием базовых возможностей технологии CUDA или библиотеки примитивов глубокого обучения [cuDNN](https://developer.nvidia.com/cuDNN).

Разработчики Caffe поддерживают возможности создания, обучения и тестирования полносвязнных и сверточных нейросетей. Входные данные и преобразования описываются понятием слоя. В зависимости от формата хранения могут использоваться следующие типы слоев исходных данных:

* DATA – определяет слой данных в формате leveldb и lmdb;
* HDF5\_DATA – слой данных в формате hdf5;
* IMAGE\_DATA – простой формат, который предполагает, что в файле приведен список изображений с указанием метки класса.

Преобразования могут быть заданы с помощью слоев:

* INNER\_PRODUCT – полностью связанный слой;
* CONVOLUTION – сверточный слой;
* POOLING – слой пространственного объединения;
* Local Response Normalization (LRN) — слой локальной нормализации.

Наряду с этим, при формировании преобразований могут использоваться различные функции активации [6].

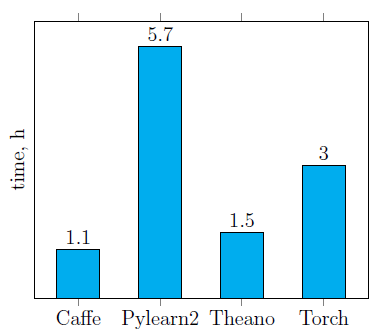


Рисунок 1.8 – Сравнение времени обучения выбранными библиотеками свёрточных сетей. GPU реализация.

**1.5** Выбор средства для реализации внешнего интерфейса сервера

Наиболее простыми и популярными Python-фреймворками для работы с сетью являются Bottle и Flask. Ниже подробнее рассматриваются эти 2 фреймворка.

[Bottle](http://bottlepy.org/) – это быстрый, простой и легкий микро веб-фреймворк для Python. Он распространяется в виде одного файла-модуля и не имеет никаких зависимостей, кроме стандартной библиотеки Python. Данный фреймворк требует установленный Python версии 2.5 или выше. Как было сказано выше Bottle не имеет дополнительных зависимостей, что упрощает процесс установки. Для установки Bottle можно воспользоваться easy\_install от Python [7]. Преимуществом данной библиотеки является скорость разработки небольших проектов.

Flask – [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для создания [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на языке программирования [Python](https://ru.wikipedia.org/wiki/Python), использующий набор инструментов Werkzeug, а также шаблонизатор [Jinja2](https://ru.wikipedia.org/wiki/Jinja). Как и Bottle Flask относится к категории так называемых [микрофреймворков](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA&action=edit&redlink=1) – минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности. Поддерживается установка посредством пакетного менеджера [PyPI](https://ru.wikipedia.org/wiki/PyPI). Flask требует версию Python 2.6 либо более высокую [8]. Пример простейшего веб-приложения с использованием данного фреймворка:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

return "Hello World!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

В процессе обзора данных фреймворков не был сделан однозначный выбор. Несмотря на простоту Bottle относительно Flask, сложно предугадать, окажется ли достаточно того функционала, что предоставляет эта библиотека, для реализации поставленной задачи.

Было решено произвести окончательный выбор фреймворка в процессе функционального проектирования, когда будет определён тот минимальный интерфейс, необходимый для передачи исходных и обработанных изображений по сети.

**1.6** Обзор ПО Docker

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в среде [виртуализации на уровне операционной системы](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B). Позволяет «упаковать» приложение со всем его [окружением](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) и зависимостями в контейнер, который может быть перенесён на любую [Linux](https://ru.wikipedia.org/wiki/Linux)-систему с поддержкой [cgroups](https://ru.wikipedia.org/wiki/Cgroups) в [ядре](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), а также предоставляет среду по управлению контейнерами.

Для экономии дискового пространства проект использует файловую систему [Aufs](https://ru.wikipedia.org/wiki/Aufs) с поддержкой технологии [каскадно-объединённого монтирования](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%81%D0%BA%D0%B0%D0%B4%D0%BD%D0%BE-%D0%BE%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5): контейнеры используют образ базовой операционной системы, а изменения записываются в отдельную область. Также поддерживается размещение контейнеров в файловой системе [Btrfs](https://ru.wikipedia.org/wiki/Btrfs) с включённым режимом [копирования при записи](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%B8_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B8).

В состав программных средств входит [демон](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)) – [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) контейнеров, [клиентские](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) средства, позволяющие из [интерфейса командной строки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8) управлять образами и контейнерами, а также [API](https://ru.wikipedia.org/wiki/API), позволяющий в стиле [REST](https://ru.wikipedia.org/wiki/REST) управлять контейнерами программно.

Демон обеспечивает полную изоляцию запускаемых на узле контейнеров на уровне файловой системы (у каждого контейнера [собственная корневая файловая система](https://ru.wikipedia.org/wiki/Chroot)), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами libcontainer), на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имён и соответствующим виртуальным сетевым интерфейсам).

Набор клиентских средств позволяет запускать процессы в новых контейнерах, останавливать и запускать контейнеры, приостанавливать и возобновлять процессы в контейнерах. Серия команд позволяет осуществлять мониторинг запущенных процессов. Новые образы возможно создавать из специального сценарного файла, возможно записать все изменения, сделанные в контейнере в новый образ. Кроме того, в интерфейсе командной строки встроены возможности по взаимодействию с публичным репозиторием Docker Hub, в котором размещены предварительно собранные образы контейнеров. Например, команда docker search позволяет осуществить поиск среди размещённых в нём образов. Образы можно скачивать в локальную систему, возможно также отправить локально собранные образы в Docker Hub [9].

Было решено использовать Docker, т. к. это удобное средство, с помощью которого, в дальнейшем можно будет с минимальными усилиями разместить готовый сервис на серверной машине с Linux системой и предустановленным ПО Docker.

**1.7** Обзор Android SDK

В качестве клиентской части было выбрано приложение под ОС Android. Клиент должен иметь возможность поставлять полутоновые изображения на сервер. Если применять программное средство для реставрации устаревших фотографий или кинолент, необходимо соответствующее оборудование. Обычному прикладному приложению, без доступа к такому оборудованию, относительно сложно получить доступ к подобной фотографии. Приложение на мобильной платформе в свою очередь имеет доступ к камере мобильного телефона. Фотографии, полученные таким образом можно преобразовывать в полутоновые на клиентской стороне, а затем отправлять на сервер. Построив клиентскую часть подобным образом, можно использовать её, демонстрируя возможности нейронной сети и одновременно популяризируя эту сферу. Также причиной выбора платформы Android можно считать широкое распространение ОС и приложений по обработке изображений на этой системе, а также наличие опыта разработки программ под эту платформу.

Приложения для Android пишутся на языке программирования Java. Инструменты Android SDK компилируют написанный код – и все требуемые файлы данных и ресурсов – в файл APK – программный пакет Android, который представляет собой файл архива с расширением apk. В файле APK находится все, что требуется для работы Android-приложения, и он позволяет установить приложение на любом устройстве под управлением системы Android.

В системе существует понятие компонентов, из которых состоит приложение для Android. Компонент представляет собой отдельную точку, через которую система может войти в приложение. Не все компоненты являются точками входа для пользователя, а некоторые из них зависят друг от друга.

Существуют четыре типа компонентов [10].

* операции представляют собой один экран с пользовательским интерфейсом;
* службы представляют собой компоненты, которые работают в фоновом режиме и выполняют длительные операции, связанные с работой удаленных процессов; служба не имеет пользовательского интерфейса;
* поставщики контента управляют общим набором данных приложения; данные можно хранить в файловой системе, базе данных, в Интернете или любом другом постоянном месте хранения, к которому у приложения имеется доступ;
* приемники широковещательных сообщений представляют собой компоненты, которые реагируют на сообщения, распространяемые по всей системе; многие из этих сообщений рассылает сама система, однако есть возможность реализации собственных типов.

**1.8** Обзор средства для реализации внешнего интерфейса клиента

Для реализации внешнего интерфейса для работы клиентской части с сетью будет использована библиотека Retrofit 2. Такой выбор обусловлен удобством использования данной библиотеки, высокой скоростью работы, отличной документацией на официальном сайте, а также достаточно большим опытом работы с этой библиотекой.

Retrofit позволяет сделать полноценный REST-клиент, который может выполнять POST, GET, PUT, DELETE запросы. Для обозначения типа и других аспектов запроса используются аннотации. Например, для того, чтобы обозначить, что требуется GET запрос, необходимо написать аннотацию @[GET](https://habrahabr.ru/users/get/) перед методом, для POST запроса – @[POST](https://habrahabr.ru/users/post/), и так далее. Нотации описываются в стандартном интерфейсе языка Java [11].