МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тверской государственный университет»

Направление 02.04.02 – Фундаментальная информатика и информационные технологии

Магистерская программа «Информационные технологии в управлении и принятии решений»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема: «Сопоставление ключевых точек изображений

с помощью нейронных сетей»

Автор:

Иванов Артем Николаевич

Научный руководитель:

кандидат ф.-м. наук, доцент кафедры,

Сорокин Сергей Владимирович

Допущен к защите:

Руководитель ООП:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Язенин А.В./

(подпись, дата)

Заведующий кафедрой: информационных технологий

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Язенин А.В./

(подпись, дата)

Тверь, 2018

**Оглавление**

[Введение 4](#_Toc515981994)

[Актуальность 5](#_Toc515981995)

[Цель и задачи исследования 6](#_Toc515981996)

[Оценка состояния темы исследования 6](#_Toc515981997)

[Структура работы 7](#_Toc515981998)

[1. Задача сопоставления ключевых точек 9](#_Toc515981999)

[1.1. Определение ключевой точки и другие термины 9](#_Toc515982000)

[1.2. Постановка задачи 14](#_Toc515982001)

[1.3. Выбор методов для реализации задачи 14](#_Toc515982002)

[2. Подготовка к разработке нейронной сети 16](#_Toc515982003)

[2.1. Основные определения и термины 16](#_Toc515982004)

[2.2. Основная структура сети для решения задачи 17](#_Toc515982005)

[2.3. Генерация обучающей выборки 19](#_Toc515982006)

[2.3.1. Общие требования к выборке 19](#_Toc515982007)

[2.3.2. Первоначальные характеристики набора областей 21](#_Toc515982008)

[2.3.3. Алгоритм работы генератора вырезанных областей 25](#_Toc515982009)

[2.3.4. Все наборы использующихся вырезанных областей 26](#_Toc515982010)

[2.3.5. Генерация обучающей выборки из набора областей 28](#_Toc515982011)

[2.4. Оценка точности сопоставления точек нейронной сетью 30](#_Toc515982012)

[3. Разработка архитектуры нейронной сети 33](#_Toc515982013)

[3.1.1. Две первых архитектуры для базовой сети 33](#_Toc515982014)

[3.1.2. Первая сеть mySiameseArtificial 35](#_Toc515982015)

[3.1.3. Вторая сеть mySiameseNet1 38](#_Toc515982016)

[3.2. Обрезка нейронной сети и новый способ сопоставления точек 40](#_Toc515982017)

[3.3. Сравнение двух первых сетей 43](#_Toc515982018)

[3.4. Новая архитектура нейронной сети и её улучшения 44](#_Toc515982019)

[4. Доработка архитектуры нейронной сети 48](#_Toc515982020)

[4.1. Подход hard-mining 48](#_Toc515982021)

[4.2. Первая реализация метода hard-mining 50](#_Toc515982022)

[4.3. Результаты первого применения hard-mining 53](#_Toc515982023)

[4.4. Новые изменения сети 54](#_Toc515982024)

[4.5. Вторая реализация метода hard-mining 57](#_Toc515982025)

[4.6. Результаты второго применения метода hard-mining 59](#_Toc515982026)

[4.7. Сравнение лучшей модели сети с известными методами 60](#_Toc515982027)

[Заключение 63](#_Toc515982028)

[Список литературы 65](#_Toc515982029)

[Приложение А: Скриншоты работы программ 69](#_Toc515982030)

[Приложение Б: Исходные коды программ 73](#_Toc515982031)

[Create\_training\_set3.py 73](#_Toc515982032)

[Do\_data\_analysis.py 77](#_Toc515982033)

[net6p.py 85](#_Toc515982034)

[compareMatching.py 95](#_Toc515982035)

[hardMiningGood.py 104](#_Toc515982036)

# Введение

С самого начала своего существования человек познаёт мир вокруг него, накапливает знания и передаёт их из поколения в поколение. Много информации об окружающем мире заключается в географических картах, чертежах. С течением времени такие объекты хранения знаний менялись, изобретались новые инструменты и новые дисциплины. Людей стали интересовать более подробная и точная информация о расположении, размерах и форме окружающих объектов. Вскоре, после изобретения в 1839 году фотографии возникает наука фотограмметрия, научно-техническая дисциплина, занимающаяся процессом создания 3D-моделей из нескольких изображений одного объекта, сфотографированного с разных углов. Первым использовать фотографии для создания топографических карт впервые предложил французский геодезист [Доминик Ф. Араго](http://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B0%D0%B3%D0%BE,_%D0%94%D0%BE%D0%BC%D0%B8%D0%BD%D0%B8%D0%BA_%D0%A4%D1%80%D0%B0%D0%BD%D1%81%D1%83%D0%B0) примерно в 1840 г., а в 1860 г. французский военный инженер Э. Лосседа выполнил фотографирование Парижа с крыши высокого здания и по фотоснимкам создал план, точность которого оказалась выше плана, полученного геодезическим методом[1]. Дальнейшее развитие технологий авиа и космического строения вывели дисциплину на новый уровень детализации. Количество информации вместе с тем росло, и её обработка переносилась на технические устройства. Высокое качество фотографии и детализации дали возможность составления 3D-моделей архитектурных строений, декора зданий, которые были гораздо меньше больших объектов рельефа местности. Одними из первых решение такой задачи на компьютере предложили Марр и Поджио в 1976 году. Их итеративный алгоритм ищет наиболее близкие точки между изображениями и строит сглаженную поверхность объекта на основе сопоставленных точек[2]. Данное исследование, а также череда других формирует научную дисциплину компьютерного зрения, занимающуюся извлечением информации из стерео изображений. С совершенствованием компьютеров и средств записи видео- и фотосъёмки появляется возможность также построения траектории движения камеры по отслеживанию изменения положения точек на кадрах видеоряда. Данная задача, как и задача построения 3D-объекта, строится на основе смещения точек, «движения». Таким образом, фотограмметрия и компьютерное зрение сформировали технологию «Structure from motion» (Структура из движения). Данная работа нацелена на исследование подзадачи общей задачи технологии «Structure from motion» - сопоставлению ключевых точек изображений, а именно – решение с помощью использования нейронных сетей.

## Актуальность

Актуальность темы работы обусловлена прежде всего большими перспективами и охватом технологии «Structure from motion»:

1) используется во многих сферах деятельности: медицине, спелеологии, геологии;

2) специалистам требуется стоить трехмерные модели пещер и различных предметов местности. Существующие на сегодняшний день технологии имеют ряд недостатков. Так же показывают плохой результат работы в “специфических” условиях, к которым можно отнести резкий перепад света на последовательных кадрах;

3) при построении топологических карт местности данная технология по сравнению с аэросъемкой позволяет добиться большей точности при минимальном количестве затрат;

4) используется для оценки, мониторинга, представления и реставрации памятников культурного наследия[3].

Задача сопоставления точек имеет широкий круг применения: создание панорам, создание стереопары и реконструкция трехмерной модели объекта по его двумерным проекциям, распознавание объектов и поиск по образцу из какой-то базы, слежение за движением объекта по нескольким снимкам, реконструкция аффинных преобразований изображений.

Также, недавние исследования в области нейронных сетей, такие как: «LIFT: Learned Invariant Feature Transform» [4] показывают, что традиционные методы решения задачи сопоставления точек с помощью эвристических алгоритмов компьютерного зрения уступают в эффективности обученным моделям нейронных сетей.

Вместе с этим следует отметить, что область знаний, рассматривающая такого рода задачи (компьютерное зрение) достаточно молода, соответственно еще нет определенного универсального метода, решающего все вышеперечисленные проблемы в полном объеме, т.е. для всех входных изображений. В тоже время существующие алгоритмы для решения данных задач постепенно совершенствуются, в том числе и с применением нейронных сетей.

## Цель и задачи исследования

В данной научно-исследовательской работе поставлена цель:

* создать нейронную сеть для сопоставления ключевых точек изображений.

Для достижения данной цели необходимо выполнить ряд задач:

* ознакомиться с задачей сопоставления ключевых точек
* разработать архитектуру нейронной сети для сопоставления ключевых точек
* провести обучение и в случае необходимости доработку нейронной сети
* сравнить полученные результаты с известными методами
* оформить текст диссертации

## Оценка состояния темы исследования

Существует множество алгоритмов для сопоставления особых точек на изображениях. Самыми известными и широко используемыми являются SURF (Speeded Up Robust Features) и SIFT (Scale-invariant feature transform) [5]. Также надо упомянуть быструю и оптимизированную альтернативу предыдущим алгоритмам - Oriented FAST and rotated BRIEF (ORB). Он объединяет детектор локальных точек FAST и дескриптор BRIEF[6]. Некоторые из представленных выше алгоритмов являются запатентованными, и показывают нерезультативность работы в сложных условиях. Недавние исследования по нейронным сетям предлагают использовать алгоритм LIFT, который превосходит результаты своих аналогов SIFT и SURF[4]. Но при этом всегда есть возможность улучшить скорость обучения и структуру сети для достижения более точных показателей.

## Структура работы

Работа состоит из оглавления, введения, четырех глав, заключения, списка литературы и двух приложений.

Во введении описываются актуальность и оценка состояния темы исследования на данный момент. Ставятся цель и задачи.

В первой главе даётся подробное определение ключевой точки, постановка задачи сопоставления точек, также описываются основные термины, используемые в рассматриваемой области.

Во второй главе описывается подготовка к созданию нейронной сети, основные определения, детали создания вырезанных областей ключевых точек, алгоритмы генерации обучающей выборки и оценки точности сопоставления точек моделью сети.

Третья глава посвящена разработке архитектуры нейронной сети, здесь рассматриваются порядка десятка различных моделей, даётся их сравнительная характеристика и выбирается лучшая.

В четвертой главе рассматривается метод hard-mining для изменения обучающей выборки и повышения точности лучшей модели нейронной сети. Также приводятся новые изменения архитектуры сети и сравнение лучшей модели с известными методами.

В заключении подводятся итоги проделанной работы.

Список литературы содержит 26 наименований источников, которые использовались для изучения вопросов, поставленных для достижения цели работы.

Работа содержит два приложения, в которых приводятся скриншоты работы программ и их исходный код.

# Задача сопоставления ключевых точек

Задача данной диссертации рассматривается в контексте задачи «Structure from motion» - технологии получения трехмерных моделей предметов и карт местности по данным видео и фотосъёмки. В свою очередь её можно разделить на ряд подзадач:

1. Работа детектора: поиск точек интереса (угловых);
2. Работа дескриптора: описание характеристических особенностей найденных точек
3. Поиск соответствия между дескрипторами точек для пар изображений;
4. Фильтрация ложных соответствий;
5. Определение смещения камеры между кадрами по векторам смещения пикселов;
6. Нахождение трёхмерной структуры.

Моя работа сфокусирована на 2-ой и 3-ей подзадачах, которые и выполняют сопоставление ключевых точек.

## Определение ключевой точки и другие термины

Для задачи сопоставления точек изображений нам необходимо определить «локальные особенности» или «ключевые точки», которые должны быть инвариантными (неизменными) к различного рода преобразованиям:

- геометрическим: смещение, вращение, масштабирование;

- фотометрическим: изменение яркости, шумовое воздействие.

Для роли «особых точек» подходят контуры изображения, которые мы можем получить применением операторов Прюитта и Собеля – специальных методов компьютерного зрения, предназначенных для выделения границ. Однако, контуры инвариантны только относительно яркости изображения.

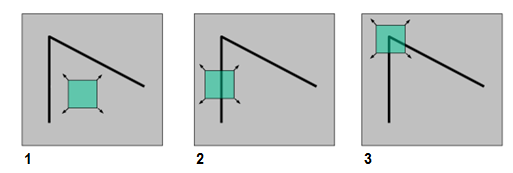


Рисунок 1. Смещение области по изображению

Рассматривая обычное смещение некоторой области по изображению (рис.1), положение, при котором минимальное смещение области вызывало бы большое изменение содержимого области не зависимо от направления.

На плоском участке 1 вовсе не наблюдается изменений, на участке 2 – нет изменений, при смещении вдоль ребра, а вот на последнем 3-ем положении можем наблюдать существенное изменение при смещении во всех направлениях. Таким образом, особыми точками изображения лучше всего выступают всевозможные угловые точки.

Математически это можно описать следующим образом:

Пусть выбранная область W – текущая область изображения, а (u,v) –координатное смещение. С помощью формулы квадрата разности мы сможем посчитать квадратичную ошибку при смещении E(u,v):

При этом, мы можем рассмотреть разложение I(x + u, y + v) в ряд Тейлора:

Так как нам нужно небольшое смещение области, то получившаяся частичная сумма ряда Тейлора без величин высшего порядка будет подходящим для нас приближением первого порядка:

Подставляя в нашу формулу, получим:

Что в свою очередь можно записать как:

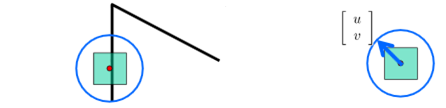


Рисунок 2. Область смещения точки по контуру

Синий круг – область малого смещения. Чтобы найти направление (вектор (u,v)), который даёт наибольшее значение E(u,v) квадратичной ошибки, нужно рассмотреть собственные векторы матрицы H=.

Собственный вектор матрицы А – это вектор x, такой что: *Ax = λx.*

Скалярное значение λ – это собственное значение, которое находится решением уравнения:

В нашем случае: А = H – матрица 2x2, поэтому имеем:

Решение:

Узнав λ, можно найти собственный вектор x, решив линейную систему:

На рис.3 показан пример области и её смещения по разным направлениям.

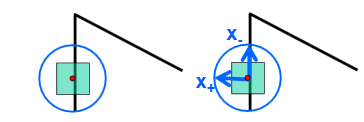


Рисунок 3. Пример области и смещения по двум направлениям

Перебирая таким образом всевозможные смещения как показано на рис.3. и вычисляя собственные вектора, получаем:

- это вектор направления наименьшего изменения значения E;

- это вектор направления наибольшего изменения значения E;

Соответственно, собственные значения для данных векторов λ- и λ+

показывают числовые значения изменения E.

Рассматривая всевозможные области на изображении с углом, получим рис.4.

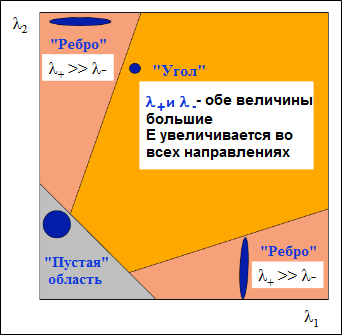


Рисунок 4. Определение типа особой точки с помощью собственных значений

Данное изображение показывает, что в случае рёбер максимальное и минимальное собственные значения сильно отличаются (одно значение >> (много больше) другого). А в случае области «угол», и наибольшее, и наименьшее *λ* оба имеют большие числовые значения. Это означает, что значение функции квадратичной ошибки увеличивается во всех направлениях[8].

Таким образом, ключевая точка — это такая точка изображения, которую можно отличить от любой другой точки изображения. На изображении можно выделить 2 типа особых точек: углы и ребра. Самым распространенным типом таких точек являются углы. Они, в отличие от ребер, не зависят от масштаба и расположения изображения. Следовательно, их можно однозначно сопоставить для пар изображений. Углы можно определить, используя различные детекторы.

Детектор — это метод получения особых точек изображения. На вход детектору подается черно-белое изображение. На выходе получается матрица, значения элементов которой определяют степень правдоподобности нахождения угла в исследуемых пикселях изображения. После выполняется отсечение участков пикселей со степенью правдоподобности, меньшей некоторого порога. Оставшиеся после отсечения точки являются особыми.

Для нахождения соответствия между точками для пар изображений к каждой угловой точке должно прилагаться её описание (дескриптор). Если дескрипторы точек на разных изображениях близки, то можно считать, что один и тот же физический объект.

Дескриптор — это небольшая окрестность вокруг ключевой точки. В идеале, дескриптор должен быть независим от масштаба и ориентации изображения. Тогда поиск соответствий сводится к обходу всех дескрипторов одного изображения и поиску наиболее близкого дескриптора на другом изображении. Например, дескриптор, продуцируемый алгоритмом SIFT – это область вокруг ключевой точки размером 16x16. Разделяется на окна 4x4, в каждом из которых вычисляется значение и направление градиента. Для достижения инвариантности к повороту: все направления поворачиваются на угол направления данной ключевой точки. В конечном счете, после ряда преобразований и нормализации, получается вектор из 128 элементов, который уже используется для сопоставления точек[9].

## Постановка задачи

Требуется разработать программный продукт для решения общего алгоритма сопоставления точек:

Пусть нам дано: два изображения, причём они являются «похожими», т.е. получены смещением, поворотом, проекцией точки, добавлением шума, изменением положения камеры или уровня освещенности.

1. С помощью известного алгоритма компьютерного зрения находим ключевые точки двух изображений
2. Далее, используя свой оригинальный алгоритм/программный инструмент получаем дескрипторы точек изображений
3. Используя заранее определенный критерий и стандартный алгоритм (или собственный) сопоставляем дескрипторы точек изображений. Здесь можно использовать как простой перебор точек, так и метод поиска k-ближайших соседей.
4. Так как каждый дескриптор соответствует своей точке изображения, то мы получаем пары сопоставленных точек двух изображений

Данный программный продукт должен работать под операционной системой Windows 7,8,10.

## Выбор методов для реализации задачи

Как было указано во введении данной работы, для решения задачи сопоставления точек будет использоваться нейронная сеть. Причем, она будет сравнивать области ключевых точек продуцируя дескрипторы данных областей.

В качестве языка программирования был выбран один из наиболее популярных для реализации задач машинного обучения – Python[7] (рис.5), среда разработки, отлично подходящая с точки зрения эффективности управления пакетами библиотек – Anaconda.

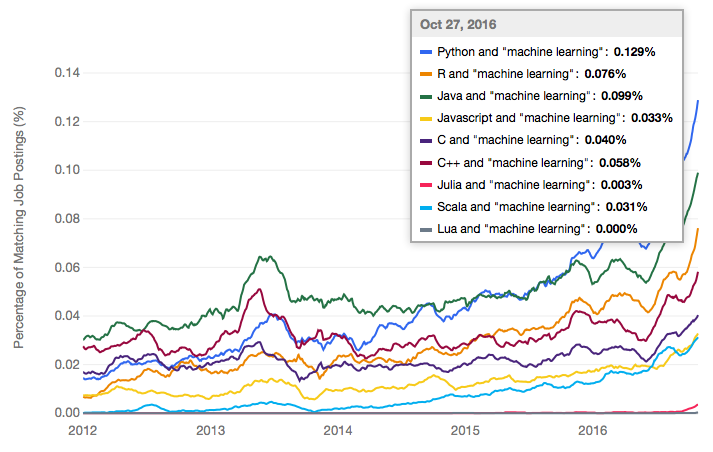


Рисунок 5. Информация, полученная с помощью поиска трендов на сайте indeed.com

Для реализации нейронной сети было решено выбрать **Keras** —открытую библиотеку, написанную на языке [Python](https://ru.wikipedia.org/wiki/Python). Она представляет собой надстройку над фреймворками [Deeplearning4j](https://ru.wikipedia.org/wiki/Deeplearning4j)[24], [TensorFlow](https://ru.wikipedia.org/wiki/TensorFlow)[25] и [Theano](https://ru.wikipedia.org/wiki/Theano)[26]. Выбор обусловлен в первую очередь компактностью и интуитивным набором абстракций, который делает простым формирования нейронных сетей не зависимо от используемой на нижнем уровне библиотеки научных вычислений.

Также использовались библиотека для обработки изображений и реализации алгоритмов компьютерного зрения cv2 (OpenCV). Она очень удобная по своему функционалу, а также уже содержит в себе алгоритмы SIFT, SURF, ORB и другие для сопоставления точек изображений, что понадобиться для моего исследования, а именно – использование детекторов особых точек изображения, получение дескрипторов и работа с областями изображений.

# Подготовка к разработке нейронной сети

## Основные определения и термины

Искусственная нейронная сеть — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы.

Первой попыткой симуляции такой работы головного мозга были предприняты У. МакКалоком и У. Питтсом в начале 40-х годов 20 века. Они сформулировали основные положения теории деятельности головного мозга. Им принадлежит идея создания единицы нейронной сети – нейрона, схематическая модель которого представлена на рис.6.



Рисунок 6. Модель нейрона

В математической интерпретации нейрон – это сумматор входных сигналов x, которые он перемножает на свои собственные веса w. После чего полученная сумма идёт на вход функции активации, которая и даёт выходной сигнал. Изначально все входные и выходное значения задумывались как бинарные. Следовательно, 0 на выходе означал, что сумма весов не преодолела некий порог и нейрон не активировался.

Совокупность данных нейронов образует сеть из параметров весов, которые регулируются в течение обучения нейронной сети: на вход ей идут данные из выборки, которые также содержат соответствующий идеальный отклик. После обработки входных данных, результат отклика сети сравнивают с идеальным откликом, указанным в обучающих данных. На этом этапе вычисляется функция потерь (loss function), которая высчитывает ошибку. После выполняется процесс обратного распространения ошибки - распространение сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы.

Вкратце, общую структуру нейронной сети можно представить, как набор слоёв – отдельных групп узлов-нейронов сети. Причем первый слой сети принимает данные на вход, а последний слой выдаёт результат предсказания сети.

## Основная структура сети для решения задачи

Обращаясь к задаче исследования, нам нужно получить нейронную сеть, которая будет сравнивать два дескриптора (описания окрестности точки изображения) и вычислять, описывает ли данный дескриптор одну и ту же точку, или же разные. Предполагаемая схема поиска соответствий представлена ниже на рис.7.

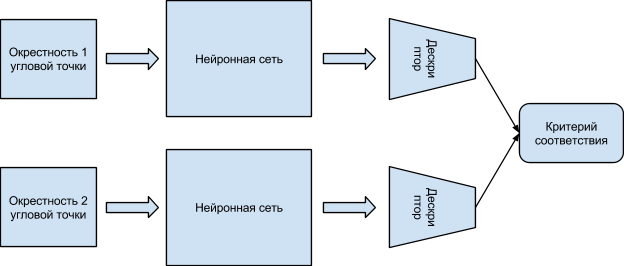


Рисунок 7. Схематическая структура сиамской сети для решения задачи

Как видим, для этой задачи лучше всего подходит модель сиамской нейронной сети – которая была впервые использована для нахождения соответствия между подписями группой ученых из AT&T Laboratories в США в 1994 году [10].

Положим для начала, как будет выглядеть общая структура (рис.8).

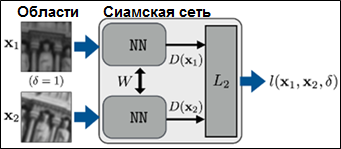


Рисунок 8 Общая структура сиамской сети для решения задачи

Как видно из рис.3, на вход подаются две окрестности точек изображения и параметр сигма, который указывает, принадлежат эти области одной и той же точке изображения (1) или разным (0). Также отметим, что отличительной особенностью такой сети является её разделение на две половинки с одинаковыми весами. В основе сиамской сети может лежать нейронная сеть любого типа и архитектуры. При этом две половинки сиамской сети объединяются критерием, с помощью которого можно определить «близость» двух дескрипторов, а соответственно и двух точек друг к другу. Таким критерием может выступать L2-норма, которая математически записывается так:

Данная норма известна как евклидова норма, которая обычно используется для вычисления расстояния между векторами. Вычисляется как разность векторов (евклидово расстояние):

## Генерация обучающей выборки

Задача разрабатываемой нейронной сети - проверять на принадлежность ключевой точке области изображения, подаваемые на вход. Как и для любой другой задачи на обучение нейронной сети главным образом влияет выбор данных, из которых сеть получает информацию. От данного набора будет зависеть не только качество сети, но и архитектура сети в зависимости от размерности подаваемой на вход выборки.

Как видим, генерация обучающих данных – задача важная и непосредственно влияющая на точность сети, поэтому в ходе исследования выборка будет претерпевать некоторые изменения. В данном пункте работы будет описан общий алгоритм формирования выборки и её первоначальные характеристики.

## Общие требования к выборке

В первую очередь отметим, что обученная сеть, хоть и должна решать задачу классификации (определять, принадлежат ли две области одной точке - класс №1 или разным – класс №0), но использоваться будет несколько иначе.

Сеть решала бы задачу классификации, если бы нас интересовало, принадлежат ли две отдельные области одной точке. А в нашем случае, согласно разделу «постановка задачи», нужно следующее:

Допустим, у нас есть область А и массив областей Bi, нам требуется, чтобы:

* 1. Сеть сравнила область А с каждой областью Bi и выдала числовые характеристики «степени близости» данных областей;
  2. На основе полученной информации с помощью алгоритма ближайших соседей или же обычного перебора можно установить наиболее ближайшую область Bi к области A, таким образом реализовав процесс сопоставления точек.

Для решения такой задачи сеть должна изучить как можно больше отличительных характеристик между областями. Реальная задача предусматривает сравнение областей двух изображений, когда выполняется сдвиг, поворот, падение яркости и т.д. Такой процесс можно реализовать выполнив определенный набор преобразований над областью и получив несколько областей, относящихся к одной точке – условимся такие пары называть «хорошими». Также можно комбинировать различные области и их преобразования, которые относятся к разным точкам, таким образом у нас получаются «плохие» пары. При этом, нужно определить алгоритм для нахождения ключевых точек на изображении, вокруг которых будут вырезаться области.

Кроме того, нужно заметить, что удобнее использовать заранее подготовленный набор вырезанных областей точек из нескольких изображений и их преобразованные аналоги. Это позволит нам выбирать, если необходимо различные комбинации областей и быстро изменять обучающую выборку на вход сети, не генерируя новые области и тем самым не выполняя лишнюю работу. Вместе с этим требуется разработать формат наименования элементов обучающей выборки, что поможет программным образом эффективно составлять комбинации областей и отличать их друг от друга, в том числе, по наименованию мы должны понимать, относятся ли области к одной точке или к разным. Здесь также есть свои требования:

1. Названия должны отличаться друг от друга

2. Названия должны давать такую информацию, что можно определить:

- две области относятся к одной и той же точке или нет;

- из какого изображения взята область. Этот параметр необходим для продуцирования «плохих» пар: сопоставление «плохих» пар различных изображений некорректно, так как существует вероятность совпадения областей по структуре рисунка, а также сам выбор детекторов определяет точки, уникальные для всего изображения – соответственно нужно сравнивать точки одного изображения.

Также есть технические параметры выборки: цвет области (имеется в виду, преобразование в ч/б или оставление цветного изображения), размер выборки и размер самих областей, причем первое и последнее непосредственно влияет на архитектуру сети и отвечает за вид входных данных.

Таким образом, для получения набора вырезанных областей нам нужно определиться со следующими характеристиками:

1. Изображения-источники, которые используются для вырезания областей;
2. Набор преобразований, который используется для расширения получаемой выборки вырезанных областей;
3. Фиксация точек: алгоритм детектора, который используется для нахождения областей на изображениях-источниках;
4. Формат наименования элементов набора вырезанных областей для эффективного генерирования обучающей выборки.
5. Общий размер выборки и кол-во ключевых точек
6. Размер вырезаемой области вокруг ключевой точки
7. Цветовые характеристики области

## Первоначальные характеристики набора областей

В соответствии с требованиями предыдущего пункта определим набор характеристик для задачи сопоставления точек и обусловим наш выбор.

1. В качестве изображений-источников выберем 16 изображений произвольной природы: кружка, коробка с кашей, коробка с хлопьями, книжки, стол, площадь, внутренний двор гостиницы, фасад дома, снимки природы, пещеры и др. Такое разнообразие поможет сети адаптироваться к всевозможного рода кадрам и лучше определять отличительные характеристики областей.
2. Основные операции для получения похожих преобразованных точек. Здесь выбор обусловлен всевозможными преобразованиями изображения, которое можно произвести:

- Вращение вокруг центра на углы 5, 10, 15 вплоть до 355 (включая) с

шагом 5 градусов. Итого получается 71 преобразование;

- Световое преобразование - изменение яркости. Выполняется,

переводом изображения из цветовой модели RGB в модель HSV (*Hue,*

*Saturation, Value* — *тон*, *насыщенность*, *значение (яркость))*,

получается изменением параметра value 4 раза на значения:

20, 40, 60, 80;

- Добавление мелкого шума. Используется функция «соль и

перец» произвольного разбрасывания черных точек;

- Масштабирование. Можно сказать, что это псевдо масштабирование,

потому как здесь используется Гауссово размытие 5x5 два раза. Идея

была взята частично из статьи про детектор SIFT[9], частью алгоритма

которого является построение масштабируемого пространства

посредством Гауссова размытия.

Таким образом на каждую найденную ключевую точку приходится 79

изображений области вокруг неё включая оригинал. Теоретически, мы

получаем не просто преобразованные области, а 78 новых

изображений, которые представлены областями и которые похожи на

оригинальное изображение.

1. Фиксация точек: Находить особые точки будем при помощи алгоритма SURF (данный алгоритм не продуцирует точки с одинаковыми координатами, в отличие от SIFT) [11]. Также, при операции поворота некоторые точки будут выходить за край рамки изображения, поэтому будем брать точки, близкие к центру изображения, относительно которого будет выполняться вращение. Т.е. из множества найденных точек будем выбирать только точки с координатами не далее, чем граница окружности с радиусом:

где h – высота – наименьший параметр размера изображения,

ar – радиус области которую берем для точки.

1. Нужно определиться с основными параметрами и кол-вом позиций под их запись в названии изображения:
2. Уникальный номер точки (не меняется при преобразованиях области вокруг точки). На изображении 1200x800 может быть обнаружено где-то до 3400 точек (рис.9), и это, если мы будем рассматривать круглую область посередине для взятия точек (как указали выше в пункте 3). Здесь не будем определять точное число позиций – гораздо проще поместить информацию о точке в начало названия области.



Рисунок 9. Найденные детектором ключевые точки изображения площади

1. Номер преобразования – здесь мы заранее знаем, что всего преобразований – 79, поэтому для номера достаточно две позиции, но возьмём 3 позиции, чтобы визуально его отделять от других значений и поместим его в конец названия.
2. Номер изображения. На начальный момент 16 изображений. Вообще говоря, это уже достаточно большое количество, с учетом того, что получается до 3400 точек с одного изображения, не считая преобразований. Аналогично номеру преобразования выделим для номера изображения 3 позиции и поместим его между номером преобразования и номером точки.

Финальный вид формата наименования представлен на рис.10.

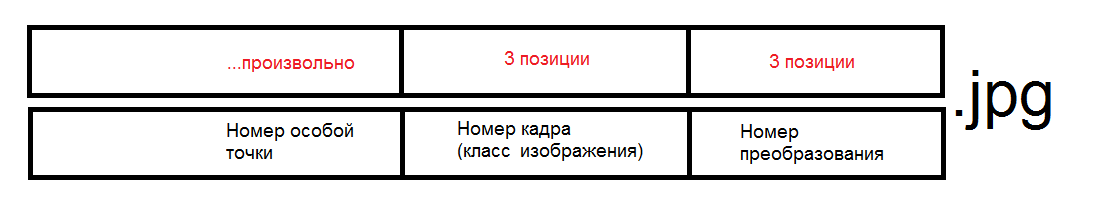


Рисунок 10 Формат наименования вырезанных областей

1. Общий размер выборки формировался произвольно. Для начала воспользовался детектором SURF, получил: 1513 точек первого изображения \* 79 преобразований = 119527 областей. Первоначально остановился на этом значении.
2. Первый размер областей выборки был 20x20 точек. Во-первых, имея компьютер небольшой мощности было решено стараться выбрать наиболее оптимальный вариант, чтобы сеть работала побыстрее. Во-вторых, некоторые базы данных для обучения нейронных сетей также содержат небольшого размера изображения и успешно на них обучаются, например, MNIST использует размер областей выборки - 28x28, а его предшественник - NIST использовал размер областей - 20x20 [12].
3. Было решено использовать цветное изображение и посмотреть, как на нём будет обучаться сеть.

Надо заметить, что характеристики набора 5-7 будут далее изменяться с рассмотрением новых сетей и введением улучшений.

## Алгоритм работы генератора вырезанных областей

1. *Выберем: N – кол-во задействованных изображений для выборки;*

*P – кол-во ключевых точек от каждого изображения;*

*R x R – размер области.*

1. *Цикл по отобранным N изображениям для генерации выборки*

*{Ik: k = 0…N}:*

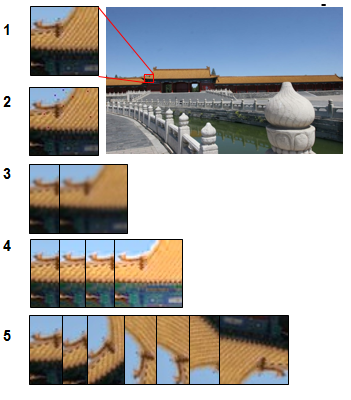
* 1. *На текущим изображении Ik находим с помощью алгоритма SURF ключевые точки {P}*
  2. *Вырезаем области R x R вокруг всех точек из P*
  3. *Цикл по точкам j = 0…кол-во\_точек(P):*
     1. *В соответствии с форматом, после каждого преобразования сохраняется новая область, относящаяся к точке j изображению k и преобразованию m:*

*<j><k><m>.jpg*

* + 1. *Неизмененная область*
    2. *Наложение шума (соль и перец)*
    3. *Масштабирование (фильтр Гаусса х2)*
    4. *Изменение яркости (х4)*
    5. *Поворот с шагом 5 градусов (от 5 до 355) (х71)*

Данная версия алгоритма – обобщенная, подробная – описана на языке программирования python в приложении Б: исходные коды программ.

Наглядно применение фильтров в цикле 4 к вырезанной области представлено на рис.7: 1 – исходная область, 2 – наложение шума, 3 – масштабирование, 4 – изменение яркости и 5 – преобразование поворота.



*Рисунок 11. Генерация вырезанного набора областей*

## Все наборы использующихся вырезанных областей

В данной работе использовалось 4 набора вырезанных областей для различных сетей. Для удобства, опишем их заранее в данном параграфе.

Как было отмечено в 2.3.2. меняться будут только характеристики, касающиеся размера, формы и цвета выборки, при этом формат наименования, фиксация точек и преобразования останутся прежними:

1. Первоначальный набор областей продуцировался только из первого изображения городской площади.

Таблица 1. Набор областей №1

|  |  |
| --- | --- |
| Изображения-источники | 1 |
| Кол-во точек с каждой картинки | 1513 |
| Размер областей | 20 x 20 |
| Цвет | Цветной (3 канала) |
| Общее кол-во элементов | 119527 |

1. Следующий набор – расширение первого. Используется больше картинок.

Таблица 2. Набор областей №2

|  |  |
| --- | --- |
| Изображения-источники | 1-8 |
| Кол-во точек с каждой картинки | Произвольно, сколько найдет детектор |
| Размер областей | 20 x 20 |
| Цвет | Цветной (3 канала) |
| Общее кол-во элементов | 968936 |

1. Необходимость следующего генерирования была вызвана низкой точностью сети и намерением улучшить её текущие показатели. В ходе визуальной проверки работы алгоритмов сравнения было замечено, что человеческому глазу довольно трудно сопоставить области одной точки размером 20x20, особенно после применения поворота как на рис.12.

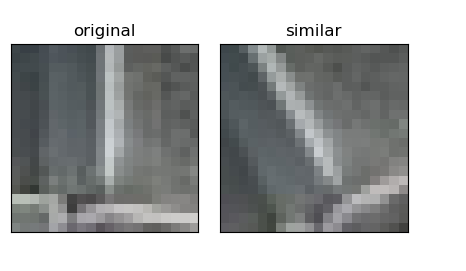


Рисунок 12 Исходная область точки и та же область, повернутая на угол

Кроме того, в других работах по изучению сопоставления точек[13] отмечается использование куда большего размера областей для обучения – 64 x 64. Следуя опять же оптимальному варианту (чтобы сеть работала быстрее) было решено выбрать размер 50 x 50.

Также данная выборка использует фиксированное кол-во точек от каждого изображения и используются все 16 изображений.

Таблица 3. Набор областей №3

|  |  |
| --- | --- |
| Изображения-источники | 1-16 |
| Кол-во точек с каждой картинки | 400 |
| Размер областей | 50 x 50 |
| Цвет | Цветной (3 канала) |
| Общее кол-во элементов | 505600 |

1. Следующий набор отличается цветом: черно-белое изображение имеет только 1 канал, что сокращает время работы сети и, как оказалось, не сильно влияет на точность обученной сети.

Таблица 4. Набор областей №4

|  |  |
| --- | --- |
| Изображения-источники | 1-16 |
| Кол-во точек с каждой картинки | 400 |
| Размер областей | 50 x 50 |
| Цвет | Ч/Б (1 канал) |
| Общее кол-во элементов | 505600 |

## Генерация обучающей выборки из набора областей

В соответствии с форматом наименования областей, пары изображений на вход сети будут формироваться в тройки и выглядеть следующим образом:

(<1001000.jpg>, <1001001.jpg>, 1) – два изображения одной области, где 1 – номер точки, 001 – номер картинки, 000 и 001 – номера преобразований и 1 – это показатель принадлежности одной точке (т.е. в данном случае принадлежит одной точке – «хорошая» пара).

Для формирования «плохих» пар аналогично, только с указанием другого показателя принадлежности – 0 (области принадлежат разным точкам). (<2001000.jpg>, <1001001.jpg>, 0) – два изображения областей разных точек, т.к. 2 != 1.

Перед обучением нейронной сети указываем, сколько требуется областей Size – общий размер, из каких изображений N, по сколько точек P от каждого изображения.

Далее алгоритм разбивает множество отобранных областей на группы таки образом, что каждая группа имеет название:

<номер точки><номер изображения>

А содержимое группы – это номера преобразований, которые относятся к данной группе.

1. Продуцирование «хороших» троек (пар) происходит перебором пар таким образом:

*Для каждой группы G из всех групп:*

*Для i=1 с шагом k=1 до T=79 делаем:*

*Продуцируем «хорошую» тройку:*

*(<G><преобразование 0>, <G><преобразование i>, 1)*

1. Продуцирование «плохих» троек происходит произвольным подбором пар областей из разных групп, таких что у них в названии совпадают номера изображений, но не совпадают номера точек.

При этом, для корректного обучения сети создаётся одинаковое кол-во «хороших» и «плохих» пар.

Так как обучающая выборка играет большую роль для архитектуры и обучения сети, и в процессе исследования будет рассмотрено несколько видов сетей, то всюду далее будем указывать основные параметры используемой обучающей выборки таким образом:

1. Номер набора вырезанных областей

(из которого формируется выборка)

1. Общее кол-во элементов выборки (обучение+тест)
2. Кол-во используемых изображений
3. Кол-во используемых точек от каждого изображения

## Оценка точности сопоставления точек нейронной сетью

Для оценки точности было решено остановиться на одном из самых сложных преобразований - поворота. Положим его достаточным для определения показателя точности сопоставления точек обученной нейронной сетью.

Идея алгоритма заключается в сопоставлении точек, взятых из исходного изображения и изображения повернутого на заранее заданный угол. Таким образом, в результате получив пары точек, мы сможем определить точность сопоставления, для этого нужно всего лишь найти новое положение первой точки, применив преобразование поворота.

Алгоритм сопоставления точек:

1. ***Первоначальные данные:***

*i - номер изображения для сопоставления точек*

*p – кол-во точек для сопоставления (для изображения i и для измененного изображения i)*

*d – угол поворота изображения*

1. ***Взять два изображения:***

*Картинка 1: Оригинальное изображение i*

*Картинка 2: Повернутая картинка 1 на угол d*

1. ***Найти координаты p ключевых точек картинки 1 (SURF)***
2. ***Найти координаты p ключевых точек картинки 2 (Поворотом точек из пункта 2)***
3. ***Вырезать по координатам изображения областей из картинок***

*Для картинки 1:*

*Mas1 = [img0, img1…] – причём индекс изображения в массиве mas1 совпадает с индексом координат этого изображения на картинке 1.*

*Аналогично для картинки 2*

*Заметим, что соответствующие друг другу изображения областей будут иметь одинаковое значение индекса.*

1. ***Сопоставить точки с помощью нейронной сети с помощью любого подходящего алгоритма (поочередно подавая пары на вход)***

*Получить массив пар индексов изображений: [(индекс из mas1, индекс из mas2) ….]*

*Затем проверить точность сопоставления, согласно замечанию из п.5, сопоставление правильное, если значения индексов равны.*

1. ***Выбрать несколько пар индексов и проиллюстрировать соответствия***

Данный алгоритм является обобщенным. Подробные его версии содержаться в Приложении Б: Исходные коды программ. Также, абстрактное представление пункта 6 обусловлено тем, что для разных моделей сетей поиск ближайших областей будет выполняться по-разному. Например, если мы рассматриваем сеть, которая выдаёт вероятность принадлежности классу (1/0), то нам нужно наибольшее значении вероятности (максимум), что это области относятся к одной точке. Наоборот, если сеть выдаёт числовое значение расстояния (насколько различаются области), то нам нужно искать минимум.

Отметим, что подходящими алгоритмами для пункта 6 могут быть:

1. Обычный алгоритм перебора точек с нахождением наибольшего «показателя близости» между точками;
2. Алгоритм поиска ближайших соседей из библиотеки FLANN.

В данной работе было решено воспользоваться обычным алгоритмом перебора, т.к. такой алгоритм прост и не нуждается в настройке параметров. Таким образом, можно уделить больше внимания контролю характеристик архитектуры сети и обучающих выборок, которых довольно много, как можно узнать из параграфов 2.3.4. и 2.3.3.

# Разработка архитектуры нейронной сети

Получив простое начальное представление модели сети в разделе 2.2, нужно настроить её параметры и работу. Первым из таких параметров выступает половинка сиамской сети – базовая нейронная сеть.

## Две первых архитектуры для базовой сети

Для начала были рассмотрены две модели для сравнения, взятые из разных источников (представлены на рис. 13).

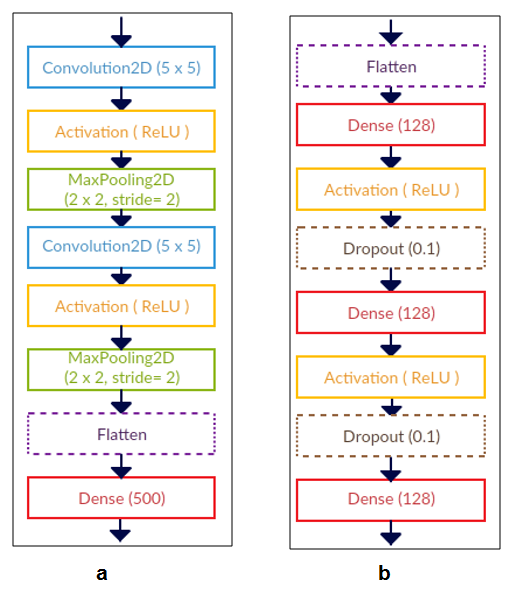


Рисунок 13 Первоначальные архитектуры базовой нейронной сети

На рис.13. первая модель «а» – это сверточная нейронная сеть, изначально решала близкую задачу – определяла похожие изображения (места отдыха). Была представлена программистом из Сан-Франциско Суджит Палом [14]. Состоит из 8 слоёв:

Convolution2D – слой свёртки, выполняет преобразования схожие с оконным преобразованием фильтра: проходит по всей матрице изображения и перемножает пиксели вокруг точки на веса, затем получает суммарный отклик всех перемноженных величин. При этом веса данного фильтра меняются в процессе обучения сети.

Свёртки представляют собой 3D тензоры (многомерные матрицы), которые имеют две пространственные оси длины и ширины, а также оси глубины (RGB) – отвечают за цвет.

Операция свертки извлекает все участки (patch – область, размером с окно фильтра) входных данных и применяет к каждому одно и тоже преобразование, получая на выходе карту признаков, которая также представляется 3d-тензором с длиной и шириной, только за ось глубины выступает уже не цвет, а фильтры – к примеру, на более высоком уровне, такая ось будет показывать количественную характеристику признака «наличие лица на изображении». В итоге получается карта признаков, которая похожа на оригинальное изображение, только теперь вместо цвета мы получаем «отклик» фильтра в данной точке и соответственно исходные входные данные уменьшаются в размере (в зависимости от размера фильтра)[15].

MaxPooling2D - слой субдискретизации, он выполняет операцию максимума на тензоре (многомерной матрице), в данном случае – матрице изображения, получая максимальное значение из всех значений взятых из области, определяемой размером окна этого слоя. Например, MaxPooling2D (2x2, stride =2) имеет окно размером 2x2, соответственно берёт максимум 4-х значений рассматриваемой области 2x2.

Параметр stride - это шаг окна субдискретизации и свёртки, который позволяет обрабатывать не все пиксели изображения, а пропускать их часть, что не даёт сети создавать слишком много параметров для обучения и оптимизирует время её работы.

Вторая структура сети «b» является простой, состоит из Dense-полносвязных слоёв. Наглядно их можно представить, как переплетение выходов всех нейронов предыдущего слоя с входами нейронами последующего (рис.14).

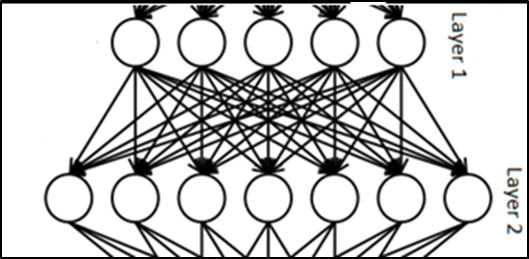


Рисунок 14 Полносвязный слой Layer 2

Также здесь используется слои Dropout – они защищают сеть с такой переплетенной структурой от переобучения, отбрасывая часть используемых нейронов сети.

Выбрав две первые архитектуры для базовых нейронных сетей нужно было закончить построение двух сиамских сетей и сравнить результаты их работы.

## Первая сеть mySiameseArtificial

За основу первой сети было решено взять архитектуру первой сети из статьи про предсказание похожести изображений [14] разработчика ПО из Сан-Франциско, Суджита Пала (структура базовой сети указана на рис.13. а). Он изучал задачу сопоставления «похожих» изображений из готовой выборки INRIA Holidays Dataset. Исходные входные размеры изображений для его сети были 224 x 224, цветные. Суджит получил небольшие результаты точности сети – около 60% при обучении, поэтому продолжил эксперименты с построением сетей из keras.zoo – библиотеки с уже натренированными моделями сетей, тем самым реализовав метод “transfer learning” – переносное обучение. Данный подход использует готовые сети для получения векторов, объединяет их и затем прогоняет через классификатор (рис. 15).

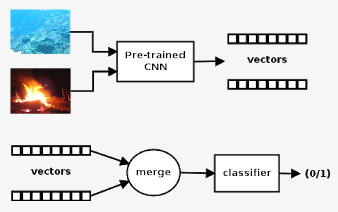


Рисунок 15. Сеть, реализованная при помощи "transfer learning"

Таким образом, он достиг лучших результатов точности 96.5%, получив вектора с помощью натренированной сети Inception (Google), далее объединив их скалярным произведением и использовав в конце классификатор XGBoost.

Несмотря на такой результат подхода «transfer learning», было решено использовать первый подход Суджита с обычной реализацией сиамской нейронной сети. Архитектура его сети была изменена для работы с нашей продуцированной выборкой. Для подробности опишем особенности реализации сети:

Во-первых, сеть решает задачу классификации. У нас есть класс – 0

(области принадлежат разным точкам) и класс 1 (когда области принадлежат одной точке. Причем на выходе сеть выдаёт вектор из двух элементов: вероятность класса 0 и вероятность класса 1. Этим обусловлен выбор функции потерь – “categorical\_crossentropy”.

Другой особенностью является использование пяти слоёв после объединения половинок сиамской сети и таким образом формирование сложного полносвязного блока из Dense, Dropout и Activation слоёв. Последним слоем идёт функция мягкого максимума (Softmax). Это стандартное её положение для сети-классификатора, определяющей вероятность отнесения объекта к классу (в данном случае объект – два изображения, подающихся на вход).

В качестве половинки сиамской сети используется сверточная нейронная сеть, которая имеет два характерных свойства:

1. Зависимости (шаблоны), которым она обучается, инвариантны относительно переноса (translation invariant): если какая-то характерная особенность изображения переместиться в другой его угол, то сверточная сеть без труда определит её;
2. Сверточные сети также могут обучиться «пространственной многоуровневости зависимостей» (spatial hierarchies of patterns): т.е. на первом слое, они могут распознавать углы, ребра, пятна, следующий слой: комбинации из рёбер и углов – многоугольники, далее – узоры, и т.д.

Также, в качестве операции объединения половинок сиамской сети использовалась функция cosine\_distance, которая агрегирует два вектора в один, поэлементно перемножая их элементы.

Отметим, что было обучено 4 модели архитектуры нейронной сети mySiameseArtificial. Также была разработана программа для проверки точности сопоставления точек с помощью данных моделей. Надо заметить, что так как сеть выдаёт вероятность принадлежности двух областей к классу 1, то нас интересует наибольшая вероятность. Т.е. алгоритм сопоставления точек, описанный в 2.4 будет строится на основе поиска пар областей с наибольшей вероятностью отнесения к классу 1 (области относятся к одной точке).

Подробности обучения и результатов сопоставления точек моделями описаны в таблице 5. Процесс сопоставления точек будем проводить по картинке с пещерой на 400 точках с поворотом картинки на 50 градусов (рис.16).



Рисунок 16. Исходные изображений для обучения сети

Таблица 5. Характеристики моделей сети mySiameseArtificial

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Название натренированной  Модели сиамской сети | Набор областей для генерации обучающей выборки | Общий размер обучающей выборки | Кол-во элементов на обучение | Кол-во элементов на тест | Кол-во  часов обучения | Точность сопоставления точек (400) |
| mySiameseArtificial\_model.h5 | №1 | 239054 | 167338 | 71716 | Ок. 2 ч | 13.00% |
| mySiameseArtificial\_model\_V1.h5 | №2 | 900000 | 630K | 270K | Ок. 8 ч | 24.75% |
| mySiameseArtificial\_model\_V2.h5 | №1 | 200000 | 140K | 60K | Ок. 2 ч | 13.50% |
| mySiameseArtificial\_model\_V3h5 | №1 | 100000 | 70K | 30K | Ок. 1 ч | 7.50% |

## Вторая сеть mySiameseNet1

За основу второй сети взяли архитектуру из примеров по keras[16]. Данная сеть решает задачу похожести двух изображений цифр из готового набора данных библиотеки MNIST. Вид базовой нейронной сети указан на рис.13 b. Общая структура сети также была изменена для использования с данными моей обучающей выборки.

Главной особенностью сети является то, что она решает задачу регрессии, т.е. на выходе получается не вероятность принадлежности классу, а числовое значение «близости» первого изображения к второму. При этом используется пороговое значение 0.5, с которым сравнивается итоговый результат предсказания сети:

1. Если предсказание <0.5, то области относятся к одной точке (или же классу 0);
2. Если предсказание >= 0.5, то области – относятся к разным точкам.

Вместе с этим, для этой сети нельзя использовать в качестве функции потерь категориальную перекрестную энтропию, как это было сделано для сети mySiameseArtificial, так как она используется только для задачи классификации с четким указанием классов, вероятности принадлежности к которым сеть выдаёт на выходе. Здесь же используется функция «contrastive loss», предложенная французским ученым Яном Лекуном[17]:

Где - векторы-дескрипторы областей 1 и 2 соответственно,

Y – 0/1 – показывает, похожие ли эти области или нет,

– предсказание сети в виде расстояния,

m > 0 – порог преобразования.

Также, в качестве функции объединения половинок используется L2-норма (евклидово расстояние). Слой с данной функцией является последним и возвращает число от 0 до 1.8 – результат сети.

Отметим, что было обучено 2 модели архитектуры нейронной сети mySiameseNet1. Также была разработана программа для проверки точности сопоставления точек с помощью данных моделей. Надо заметить, что так как сеть выдаёт числовое значение и области относятся к одной точке, если оно меньше 0.5, то нас интересует пара областей, для которой сеть будет выдавать наименьшее значение. Т.е. алгоритм сопоставления точек, описанный в 2.4 будет строится на основе поиска пар областей с наименьшим значением предсказания.

Обучающая выборка для данных моделей была сгенерирована из первого набора вырезанных областей (см.2.3.4), соответственно использовалось только 1 изображение и 1513 найденных ключевых точек. Подробности обучения и результатов сопоставления точек моделями описаны в таблице 6. Процесс сопоставления точек будем проводить по картинке с пещерой на 400 точках с поворотом картинки на 50 градусов.

Таблица 6. Характеристики моделей сети mySiameseNet1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Название натренированной  Модели сиамской сети | Набор областей для генерации обучающей выборки | Общий размер обучающей выборки | Кол-во элементов на обучение | Кол-во элементов на тест | Кол-во  часов обучения | Точность сопоставления точек (400) |
| mySiameseNet1\_model\_V0.h5 | №1 | 100000 | 70K | 30K | Ок. 2 ч | 16.75% |
| mySiameseNet1\_model\_V1.h5 | №1 | 239054 | 167338 | 71716 | Ок. 8 ч | 20.75% |

## Обрезка нейронной сети и новый способ сопоставления точек

Так как результаты сопоставления точек оказались довольно низкими, то решено было использовать модели сетей по-другому: вырезать половинку сиамской сети и получить модели, которые продуцируют дескрипторы. Далее будет описано, как выполнить это обрезание на примере модели mySiameseArtificial\_model.h5 с помощью библиотеки keras.

Для начала загрузим модель:

model = load\_model('mySiameseArtificial\_model.h5')

Далее, распечатаем и подробно изучим структуру и последовательность её слоёв (рис.17).

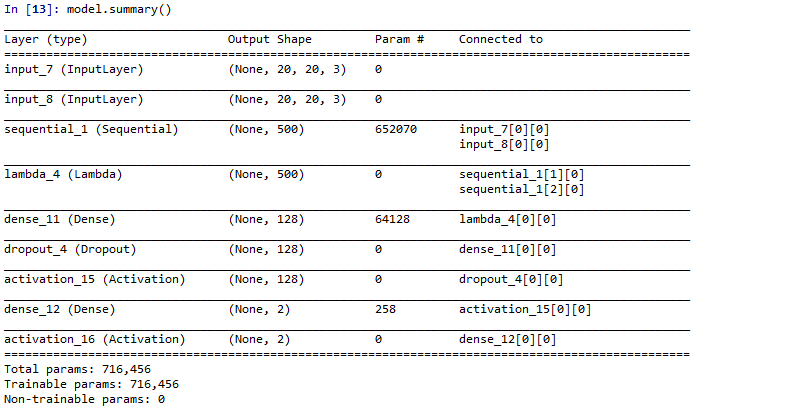


Рисунок 17. Архитектура mySiameseArtificial\_model

Требуется отрезать половину этой модели (для получения дескриптора по одному изображению). Как видим, слои lambda\_4 и sequential\_1 соединены с двумя узлами. Рассмотрим их по порядку:

Выход lambda\_4 ожидаемо даёт вектор на 500 элементов (рис.18):



Рисунок 18. Слой объединения половинок сиамской сети - lambda

Если обратиться к коду построения сети, то в этом месте мы объединяем два вектора:

distance = Lambda(cosine\_distance, output\_shape=cosine\_distance\_output\_shape)([vector\_left, vector\_right])

Спускаемся дальше и находим слой sequential\_1 – этот слой отвечает за базовые половинки нашей нейронной сети и даёт сразу три подсоединенных узла (рис.19).

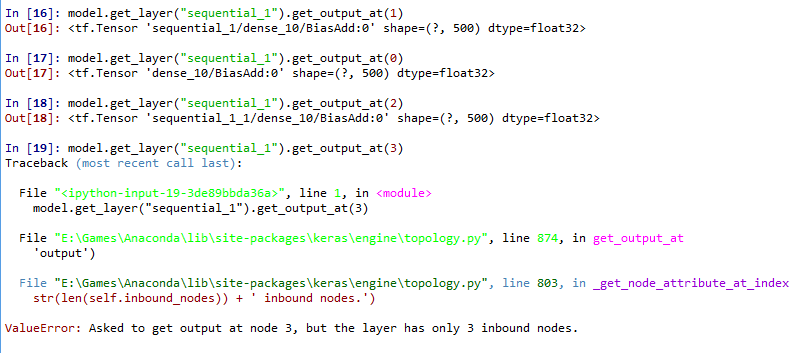


Рисунок 19. Выходы слоя sequential

Но поскольку общая структура сети указывает номера узлов 1 и 2 как выходы sequential (рис.20), то они и являются выходами половинок сиамской сети.

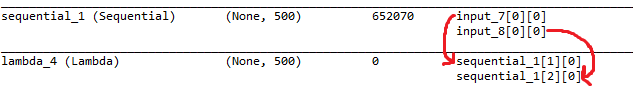


Рисунок 20. Связь слоёв sequential и lambda

В таком случае, уточнив индекс слоя sequential (в данном случае – 1), выполняем обрезание сохраняем новую модель:

new\_model = Model(input=model.input[0],

output=model.get\_layer("sequential\_1").get\_output\_at(1))

new\_model.save(‘choppedModel.h5’)

Как показывает структура обрезанной модели (рис.21) – все натренированные параметры сохранились, а модель возвращает вектор-дескриптор области изображения на 500 элементов.

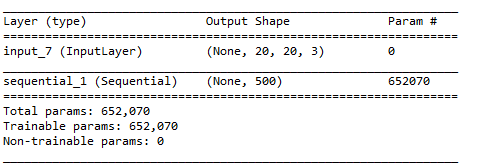


Рисунок 21. Структура обрезанной модели сети mySiameseArtificial

Отметим, что таким способом можно обрезать все остальные сети, кроме того – пропадает путаница в оценке точности сопоставления точек. Теперь не нужно использовать разные способы сравнения для первой и второй сети. Получив дескрипторы, прогоном областей двух изображений, мы сможем сделать сопоставление при помощи L2-нормы. Можно сказать, что мы будем считать евклидово расстояние между областями и находить наименьшее – таким образом сопоставлять точки двух изображений. Новый алгоритм сопоставления точек подробно описан в программном виде в Приложении Б: Исходные коды программ.

## Сравнение двух первых сетей

Таблица 7. Сравнение реализаций сетей mySiameseArtificial и mySiameseNet1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Характеристика** | **mySiameseArtificial** | **mySiameseNet1** |
| 1 | Функция потерь | "categorical\_crossentropy" (библиотечная функция) | contrastive\_loss (объявлена вручную) |
| 2 | Оптимизатор | "adam" (библиотечная) | Rms (из библиотеки keras.optimizers.RMSprop) |
| 3 | Метрика точности | "accuracy" (библиотечная) | Accuracy (объявлена вручную) |
| 4 | Кол-во слоёв | 9 (5 слоёв ушли на дополнительные преобразования после объединения векторов) | 4 |
| 5 | Формат предсказания | Вектор: [вероятность несовпадения, вероятность совпадения]  Пример: [0.34, 0.66] – точки совпадают,  [0.66, 0.34] – точки не совпадают | Число: От 0 до 1.6 \*  Если число < 0.5, то совпадение, иначе - нет  \*заключение сделано по оценке работы сети на выборке MNIST  Пример: 0.34 – точки совпадают, 0.66 – точки не совпали |
| 6 | Продолжительность обучения на выборке 100000 эл-тов | Около 1 часа | Менее 30 минут |
| 7 | Функция объединения векторов (слой lambda) | Cosine\_distance – поэлементное перемножение | euclidean\_distance |

Теперь, обрежем все полученные модели и с их помощью выполним сопоставление точек, пользуясь метрикой L2-нормы. Результаты представлен в таблице 8.

Таблица 8. Сравнение точности сопоставления точек обрезанными моделями

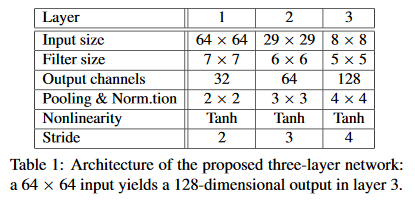
|  |  |  |
| --- | --- | --- |
| **Используемая модель сети:**  **Название (размер обучающей выборки)** | **Тип структуры сети** | **Точность сопоставления на 400 точках первого и второго изображений** |
| choppedmySiameseArtificial\_model(239054).h5 | а | 31.800% |
| choppedmySiameseArtificial\_model\_V1(900000).h5 | а | 48.800% |
| choppedmySiameseArtificial\_model\_V2(200000).h5 | а | 34.200% |
| choppedmySiameseArtificial\_model\_V3(100000).h5 | а | 32.600% |
| choppedMySiameseNet1\_model\_V0(100000).h5 | b | 22.600% |
| choppedMySiameseNet1\_model\_V1(239054).h5 | b | 7.000% |

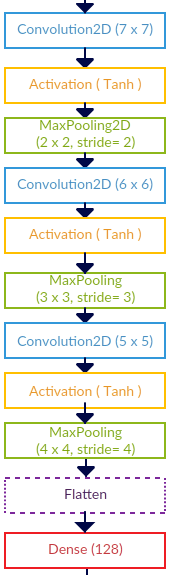
Как видим из таблицы 1, лучше всего справилась с задачей модель сети choppedmySiameseArtificial\_model\_V1(900000) со структурой базовой сети из рис.13 «а». Соответственно, можно сделать вывод, что сиамская сеть на основе сверточных сетей справилась с задачей лучше, чем другая сеть – mySiameseNet1, в основе которой последовательность простых полносвязных слоёв. Надо отметить также, что подход с обрезанием моделей позволяет легко сравнивать модели разных сетей и даёт более высокую точность.

Однако, результат сопоставления оказался небольшой. И это при использовании 900К элементов выборки и 8 часовом обучении! Поэтому далее разрабатывалась новая архитектура нейронной сети.

## Новая архитектура нейронной сети и её улучшения

Изучив статью по обучению «сверточного характеристического дескриптора ключевых точек» (Discriminative Learning of Deep Convolutional Feature Point Descriptors)[13], ознакомился с новой архитектурой сверточной нейронной сети. В работе также решалась задача сопоставления точек. Сразу же отмечу, что программного кода реализации нейронной сети и её обучения авторы не предоставляли, вместе с тем они использовали библиотеку Torch7 для mathlab для создания сиамской сети. Поэтому, архитектура их сети (рис.22) была переработана, чтобы её можно было реализовать на keras.



Рисунок 22. Архитектура сверточной сети из статьи [13]

Опишем далее особенности архитектуры новой сиамской сети mySiameseNet2:

1. Базовая сверточная сеть.

Основная структура базовой сети взята из статьи, однако авторы использовали выборку (а соответственно и формат входных данных) -

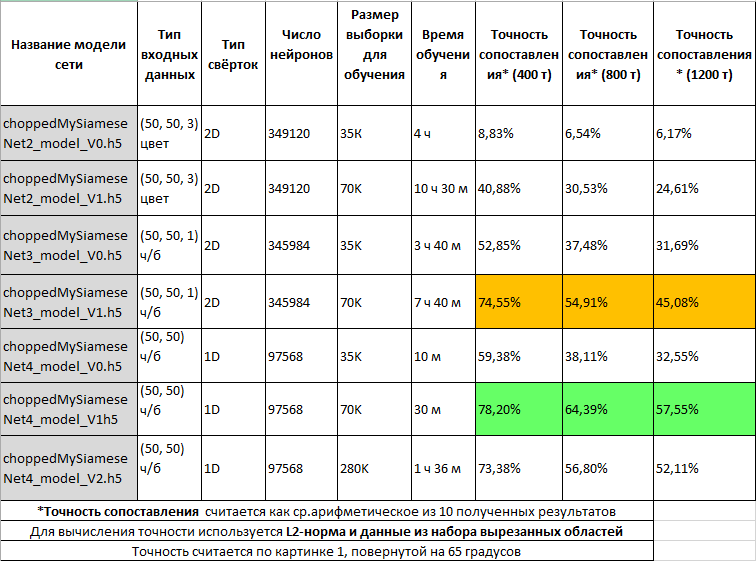
64x64. В моей же сети для генерации выборки используются наборы областей №3 и №4 с размером 50x50. Функция активации гиперболического тангенса оставлена, ссылаясь на её лучшую производительность[13]. Вместе с тем мной были опущены дополнительные слои «вычитаемой нормализации» (subtractive normalization), которая вычитает взвешенное среднее, получаемое при обходе гауссовым фильтром после первого и второго слоёв. В keras можно воспользоваться batch-нормализацией, однако, для данной задачи она понижает точность сети и увеличивает выдаваемые значения функции потерь почти до 160 (когда обычные значения функции потерь наблюдались от 0 до 1).

Рисунок 23. Базовая нейронная сеть для mySiameseNet2

Также, L2-субдискретизация была заменена субдискретизацией max-pooling (по причине отсутсвия первой в библиотеке keras);

1. Функция потерь из статьи [13] не использовалась, была оставлена contrastive loss[17];
2. Половинки сети объединяются при помощи L2-нормы;
3. Оптимизатор для расчетов обратной ошибки и градиента был оставлен прежним из билиотеки keras – Rms, в то время как авторы статьи использовали стохастический градиентный спуск.

Таблица 9. Результаты работы моделей сетей Net2-4



Далее, оставляя неизменным основную последовательность слоёв архитектуры сети mySiameseNet2, было решено поэкспериментировать с параметрами слоёв и обучающей выборкой. В результате обучили порядка 7 новых моделей на основе сетей Net2-4. Итоговая сравнительная характеристика представлена на таблице 9 и соответствующими выводами по ней:

1. Наибольшая точность при сопоставлении точек получили при использовании моделей сетей:

Net4 (choppedMySiameseNet4\_model\_V1);

Net3 (choppedMySiameseNet4\_modelV1).

1. Приём с переходом от 2D сверток к 1D сверткам уменьшил количество настариваемых параметров сетей, сократил время обучения и показал хороший прирост точности при сопоставлении точек;
2. Модели, обученные на 35К и 70К различаются по точности на 20%, следовательно, можно предположить, что увеличение обучающей выборки в 2 раза даёт прирост точности. Вместе с этим черезмерное увеличение данных для обучающей выборки, как в случае с последней моделью Net4 (280К) - даёт обратный эффект и точность понижается;
3. При увеличении числа точек для сопоставления (например, для Net2\_model\_V1) точность уменьшается.

В общем, можно заключить, что точность сопоставления точек при помощи новых моделей сетей превысила 50%, поднявшись почти до 80%. Стоит отметить, что при создании Net4 была оптимизирована работа с памятью. Для этого использовались генераторы – специальные структуры списков в python, которые после первого прочтения сразу же удаляются из памяти и не хранятся в ней полностью, как массивы данных. Это изменение способствовало уменьшению время обучения.

Такой же подход был сделан далее, для повышения точности модели сети Net3 (новая сеть Net5), при этом остальные её параметры не менялись. Однако, точность сопоставления точек снизилась на 20%, показав, что на обучение значительно влияет выбор плохих пар областей сети, который формировался каждый раз случайно.

# Доработка архитектуры нейронной сети

## Подход hard-mining

Как показало обучение сетей Net3 и Net5 предыдущего раздела, на обучение лучшей архитектуры сети влияет подбор данных обучающей выборки. Значит, из случайных вариантов генерации плохих пар существует наилучший, который позволяет добиться оптимальной точности.

Было решено подробнее рассмотреть процесс генерации обучающих пар выборки в статье «Discriminative Learning of Deep Convolutional Feature Point Descriptors»[13]. Оказалось, авторы, основываясь на наблюдении, что после определенного момента времени обучения большинство пар правильно классифицируются сетью, и прекращение их использования улучшает натренированные весы модели, предлагают стратегию агрессивной добычи «жестких» хороших и плохих пар, которая называется “hard-mining”.

В течение стадии обучения они усиливают обратное распространение ошибки обучения (the back propagation) выборки с помощью большой потери, т.е. и пары, относящиеся к одной точке (хорошие), которые были распознаны как плохие, и пары, относящиеся к разным точкам, которые были распознаны как хорошие. Этот метод показывает себя самым практичным для эффективного обучения отличительных дескрипторов (discriminative descriptors), как заявляется в статье.

Далее авторы уточняют, что данный метод – отклонение от стандартной практики. Добыча «трудноотличимых» плохих пар (mining hard negatives) – это известная процедура в контексте скользящих окон детекторов (sliding-window detectors)[18], где количество плохих экземпляров выборки буквально безгранично и всё же, большинство плохих пар легко различаются, после использования определенного количества плохих примеров для обучения. В этой статье авторы показывают, что агрессивный поиск и «трудноотличимых» хороших (hard positive), и «трудноотличимых» плохих (hard negatives) пар значительно улучшает процесс обучения. Причем, выборка, используемая в статье немного другая: она формируется из проекций 3D-точек, которые (точки) выбираются произвольно из большого множества. Авторы используют готовый набор вырезанных областей вокруг точек [19], которые расположены на картинке 1024х1024. Т.е. на каждой картинке массив 16х16 таких областей. Они уточняют, что из всего набора возможно сгенерировать 1,133,525 потенциальных хороших пар и 1 триллион потенциальных плохих пар. Такая асимметрия (имеется ввиду соотношение хороших и плохих пар – плохих пар в миллион раз больше 1,2 \* 106 / 1,12 \* 1012 ≈ 1 / 106) традиционна для решения проблем сопоставления, таких как стерео фотограмметрия и восстановления структуры движения (structure from motion) – авторы борятся с таким положением дел с помощью агрессивного поиска «трудноотличимых» плохих и хороших пар.

Из статьи можно вынести общий абстрактный алгоритм генерации выборки методом «hard-mining»:

1. Прежде сеть обучается до определенного уровня точности;
2. Далее, для каждой последующей эпохи генерируется набор плохих пар;
3. Плохие пары прогоняются через сеть и считается числовое значение функции потери;
4. Далее оставляют только самые «трудноотличимые» плохие пары, т.е. такие, которые дают меньший числовой результат предсказания сети и высокое значение потери

(в моём случае, «трудноотличимыми» плохими парами будут те, на которых сеть выдаёт результат от 0 до 0.5, соответственно, для хороших «трудноотличимых» пар моя сеть будет выдавать результат, больший 0.5);

1. Далее сеть обучают на отобранных «трудноотличимых» плохих парах.

Также, авторы добавляют, что такая процедура может быть использована и при генерации хороших пар.

## Первая реализация метода hard-mining

Далее реализуем метод поиска «трудноотличимых» пар для расширения выборки с помощью python. Для начала оценим, сколько потенциальных пар можно получить из набора №4 вырезанных областей 16 изображений.

Генерация обучающей выборки для одного изображения:

1. Генерация хороших пар:

На 1 точку приходится 79 областей (оригинал + преобразования). т.к. для нас не имеет значение порядок выбора областей, то это превращается в классическую задачу комбинаторики про сочетания:

«сколькими способами можно выбрать m из n различных предметов?»

Т.е. сколькими способами можно выбрать 2 области из 79:

На 400 точек получаем: 400 \* 3081 = 1232400 (1.2М) потенциальных пар можно получить из набора №4.

1. Генерация плохих пар:

Здесь мы будем выбирать 2 области из 400\*79=31600 (всех имеющихся областей для картинки). Однако, мы не рассматриваем области одних и тех же точек:

Выбираем первую область: 400\*79

Выбираем вторую область: 399\*79

Мы рассматриваем задачу: взять 2 области из 400 точек:

Итого 79800 сочетаний точек возможно. Теперь нужно добавить к этим

данным выбор областей для этих точек:

Домножить на 79\*79, так как дальнейший выбор не зависит дальнейший от сочетаний (79 областей одной точки и 79 областей другой отличаются друг от друга).

Итого: 79800\*79\*79 = 498031800 (498М) потенциальных плохих пар.

Так как, кол-во вариантов для поиска «трудноотличимых» пар велико, то было решено использовать потоки для распараллеливания вычислений и увеличения скорости работы. Также, весь комплекс программ был перенесен с ноутбука (на котором изначально выполнялись вычисления) на стационарный компьютер с большей мощностью процессора для сокращения времени работы программ.

**Алгоритм для “hard mining”:**

*Предварительные параметры:*

*N – кол-во точек,*

*len\_pairs - сколько хор. и плох. пар мы хотим получить*

*index\_pict - номер картинки*

*dir – директория, где лежат области*

*handleArea(name) – функция загружает область name и конвертирует её в формат входных*

*данных модели нейронной сети*

*GoodL – список хороших пар типа: <val, (img1, img2, 1)>*

*BadL – список плохих пар: <val, (img1, img2, 0)>*

*Алгоритм:*

1. *Загружаем модель нейронной сети*

*#Возможные хорошие пары:*

1. ***Для каждой точки*** *I из N точек* ***делаем****:*
   1. ***Для каждой области*** *ia1 = 0* ***с шагом*** *1* ***до*** *78(включая)* ***делаем****:*
      1. ***Для каждой области*** *ia2 = ia1+1* ***с шагом*** *1* ***до*** *78(включая)* ***делаем****:*
         1. *Загружаем img1 = handleArea(<i><index\_pict><ia1>.jpg)*
         2. *Загружаем img2 = handleArea(<i><index\_pict><ia2>.jpg)*
         3. *Получаем значение val, запуская модель сети на параметрах:*

*(img1, img2, 1)*

* + - 1. ***Если*** *val < 0.5* ***то****:*

*Переходим к следующей итерации цикла 2.1.1.*

* + - 1. *Идём по списку GoodL (с конца): j = len(GoodL)-1; j >= 0; j--*

*По очереди сравниваем значение val со значениями*

*других пар из GoodL:*

***Если*** *val > GoodL[j].val* ***то****:*

***Если*** *мы рассматриваем первый элемент (j == 0)* ***то****:*

*вставляем <(img1, img2, 1), val> в начало*

*выход из цикла по GoodL*

***Иначе:***

*переходим на новую итерацию*

***Если*** *val <= GoodL[j].val* ***то****:*

***Если*** *мы рассматриваем последний элемент (j == len(GoodL)-1* ***и*** *длина списка ещё не len\_pairs* ***то****:*

*вставляем <(img1, img2, 1), val> в конец*

*выход из цикла по GoodL*

***Иначе:***

*Вставляем <(img1, img2, 1), val> на позицию j*

*выход из цикла по GoodL*

* + - 1. ***Если*** *длина списка стала больше len\_pairs* ***то****:*

*удаляем последний элемент списка*

*#Возможные плохие пары:*

1. ***Для каждой точки*** *I= 0* ***с шагом*** *1* ***до*** *N-1(включая)* ***делаем****:*
   1. ***Для каждой точки*** *J=I+1* ***с шагом*** *1* ***до*** *N-1(включая)* ***делаем:***
      1. ***Для каждой области*** *ia1 точки I* ***делаем****:*
         1. ***Для каждой области*** *ia2 точки J* ***делаем****:*
            1. *Загружаем img1 = handleArea(<i><index\_pict><ia1>.jpg)*
            2. *Загружаем img2 = handleArea(<i><index\_pict><ia2>.jpg)*
            3. *Получаем значение val, запуская модель сети на параметрах:*

*(img1, img2, 0)*

* + - * 1. ***Если*** *val > 0.5* ***то****:*

*Переходим к следующей итерации цикла 3.1.1.1.*

* + - * 1. *Идём по списку BadL (с конца): j = len(BadL)-1; j >= 0; j--*

*По очереди сравниваем значение val со значениями*

*других пар из BadL:*

***Если*** *val < BadL[j].val* ***то****:*

***Если*** *мы рассматриваем первый элемент (j == 0)* ***то****:*

*вставляем <(img1, img2, 0), val> в начало*

*выход из цикла по BadL*

***Иначе:***

*переходим на новую итерацию*

***Если*** *val >= BadL[j].val* ***то****:*

***Если*** *мы рассматриваем последний элемент (j == len(BadL)-1* ***и*** *длина списка ещё не len\_pairs* ***то****:*

*вставляем <(img1, img2, 1), val> в конец*

*выход из цикла по BadL*

***Иначе:***

*Вставляем <(img1, img2, 1), val> на позицию j*

*выход из цикла по BadL*

* + - * 1. ***Если*** *длина списка стала больше len\_pairs* ***то****:*

*удаляем последний элемент списка*

Программная версия алгоритма на языке python описана в Приложении Б: Исходные коды программ.

Для использования потоков воспользуемся простым разделением алгоритма:

1. Сначала создаются и вычисляются потоки для хороших пар:

Действия 2.1.1.1. – 2.1.1.6. разделяются по точкам:

Пусть всего потоков - P, а точек N, тогда каждому потоку

по P/N точек.

1. Затем создаются и вычисляются потоки для плохих пар:

Действия 3.1.1.1.1. – 3.1.1.1.6. разделяются по преобразованиям:

Пусть всего потоков - P, а преобразований - A, тогда каждому потоку

по P/A преобразований со всевозможными точками.

Также отметим, что программная реализация отличается от алгоритма тем, что потоки считают каждый свой список точек (хороших/плохих), а затем, по окончании работы происходит слияние списка отдельного потока с общим списком. Так как, и списки в потоках, и общий список – отсортированы, то лучше всего с задачей слияния справляется алгоритм, основанный на кучах - heap queue algorithm [21]. Слияние в таком случае выполняется за линейное время O(n).

Такое изменение было использовано, чтобы общий большой список (плохих/хороших) пар реже использовался и не создавалась большая очередь из потоков, ожидающих своего доступа на изменение.

## Результаты первого применения hard-mining

Было сразу же решено перейти на вариант поиска «трудноотличимых» пар на 100 точках (вместо 400), так как время работы на всевозможных хороших парах областей из 100 точек (всего 308100 пар) с извлечением 25К пар с максимальным значением сети заняло 1 час 15 минут, 900 Мб ОП.

Для плохих пар, т.к. даже для 100 точек элементов слишком много: 15М, по сравнению c предыдущим – 0.3М (приблизительно в 50 раз больше), запустили на 20 точках (585К пар) – получили время работы - 2 часа 14 минут.

Сгенерированная выборка из «трудноотличимых» пар сохраняется в текстовых файлах (рис.24).

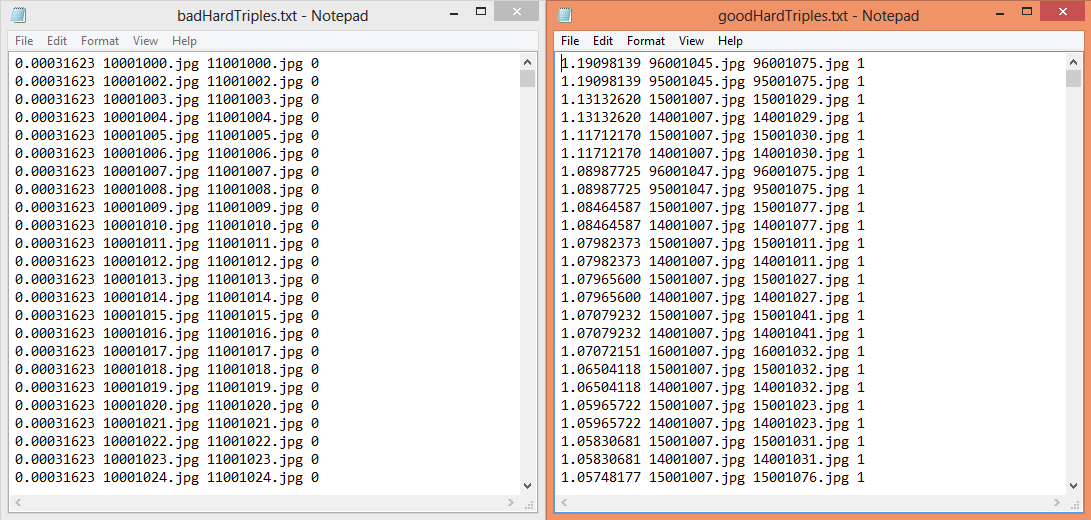


Рисунок 24. Выборка из "трудноотличимых" пар

Таким образом получили: 25К - хороших пар, 25K - плохих. Далее их объединили в общую выборку и продолжили обучать лучшую сохраненную модель – Net3 (51%). Однако, точность во время обучения (было 98%) начинает расти с 40% и останавливается на 90%. Но данная точность не относится к сопоставлению точек, на котором дообученная выдала 1-4%.

После 3-х прогонов дообучения сети на полученных данных с помощью hard-mining хороших и плохих пар 1,2,3 картинок, наблюдалась следующая динамика: 66%, 16%, 20%, 6% соответственно. Данный результат указал на наличие проблем в реализации сети и применении метода поиска «трудноотличимых» пар.

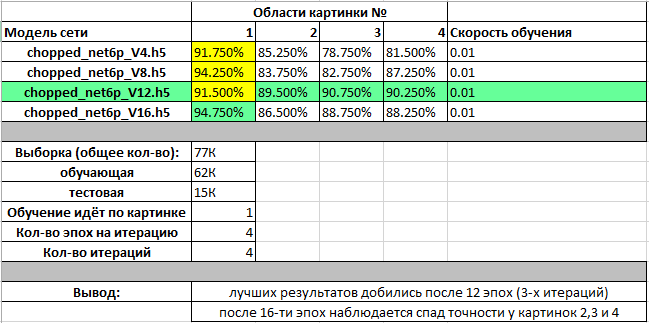
## Новые изменения сети

Сначала была решена первая проблема – это подготовка сети к дообучению её сохраненных моделей. Для этого весь процесс обучения, сопоставления точек и распечатки графиков был автоматизирован при помощи скриптов batch windows. Также для слежения за показателями потерь и точности использовался класс фиксации состояний сети, описанный здесь [22].

Далее решалась проблема изменения параметров при продолжении обучения сохраненных моделей сети. Следую предложению Deneb91 было решено заменить оптимизатор rms на sgd [23]. Несмотря на то, что rms также может продолжить обучение сохраненных моделей, sgd показал себя с лучшей стороны и увеличил точность сети на 40% без использования метода hard-mining. Так получилась новая сеть Net6p, которая отличается от Net3 оптимизатором, явным указанием скорости обучения = 0.01 и моментом = 0.9, которые были подобраны исходя из нескольких экспериментов с обучением сети. Следующая таблица 10 показывает точность сети Net6p при сопоставлении точек на 4-х картинках с подробным описанием характеристик моделей.

Надо отметить, что сеть обучалась с помощью набора №4 выборки на первой картинке и модель chopped\_net6p\_V12.h5, обучавшаяся в течение 12 эпох показала лучший результат – 90,5% в среднем на тестовых вырезанных областях 4-х различных картинок. Данная сеть была также обучена на различных точках из всех 16 картинок, однако наибольшим результатом было 83,5% точности после 16 эпох.

Таблица 10. Характеристики обучения моделей сети Net6p



Добившись такого значительного результата при использованиии SGD оптимизатора, было решено вернуться к сети Net4, которая позволила добиться наилучшей точности при сопоставлении точек (таблица 9). Заменив у данной сети оптимизатор, мы получили Net7. Было обучено две модели сети на выборках 70К и 280К пар областей ключевых точек – Net7\_V0 и Net7\_V1 соответственно.

Таблица 11. Точность сопоставления точек при помощи моделей сети Net7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Области картинки № | | | |
|  | **1** | **2** | **3** | **4** |
| chopped\_net7\_V0.h5 (70К) | 63,50% | 57,00% | 49,75% | 59,75% |
| chopped\_net7\_V1.h5 (280К) | 78,75% | 79,00% | 81,75% | 79,75% |

Однако, как видим из таблицы, результаты точности значительно уступают моделям сети Net6p, на которых в дальнейшем будет тестироваться второй подход использования hard-mining.

Вновь заметим, что ключевую роль играет подбор пар для обучающей выборки, который все еще остается произвольным, поэтому вернемся к проблеме продолжения обучения. Удостоверившись в корректности реализации автоматизированного дообучения сохраненных моделей сети Net6p c оптимизатором sgd было выяснено, что спад точности наблюдается еще при обучении и только тогда, когда мы при загрузке уже обученной модели меняем выборку (на выборку из hard-mining, например).

На рис. 26 в Приложении А: Скриншоты работы программ продемонстрированы скриншоты из двух разных дообучений сети. Под буквой «А» - продолжение обучения идёт без изменения обучающей выборки, а под буквой «В» - дообучение выполнено с заменой обучающей выборки из областей 1-ого изображения на выборку областей из 2-ого изображения. Как можно заметить под буквой «А» совершенно логично точность продолжает расти с 0.7400 до 0.7650 на первой эпохе при дообучении и при этом падает значение потерь. А вот под буквой «В» все идёт в противоположную сторону – точность падает с 0.6484 до 0.6250 и значение потерь увеличивается с 0.2174 до 0.4170.

Отсюда можно сделать вывод, что если выполнять дообучение полностью заменяя начальную выборку другой, в том числе и полученной методом hard-mining, мы не получим прироста точности, а скорее ухудшим производительность обученной модели сети.

## Вторая реализация метода hard-mining

Первое применение hard-mining не сработало из-за ряда проблем:

1. Выборка, подаваемая на вход сети могла быть плохой, т.к. плохие пары генерируются каждый раз произвольно;
2. Перебор пар для поиска «трудноотличимых» берёт области, которые не пригодятся для различения точек: то есть, могут быть области с преобразованием масштаба исходной области и преобразованием поворота исходной. Таким образом сеть обучается комбинации двух преобразований, когда нам достаточно, чтобы сеть знала, что исходная область и её преобразование относятся к одной точке;
3. И соответственно, обучение сохраненной модели сети на новой выборке, полученной методом hard-mining ухудшает её точность.

Для разрешения первой проблемы была написана программа, которая «фиксирует» выборку и сохраняет тройки в текстовый файл. Таким образом, можно отличать различные выборки и выбирать ту, на которой сеть обучается лучше.

Вторая же проблема была решена изменением алгоритма hard-mining при переборе точек. Теперь рассматриваются все возможные комбинации плохих и хороших пар с тем условием, что одна из областей пары представляет непреобразованную исходную область (где преобразование = 0, как например в ‘1003000.jpg’). Такой же подход используется при генерации исходных плохих пар для перового обучения сети.

Также, для устранения третьей проблемы, было «расширять» исходную выборку, а не заменять её полностью. И уже на расширенной выборке обучать сеть по новой. Для наглядности приведем алгоритм обучения сети с помощью hard-mining.

ОБЩИЙ АЛГОРИТМ ОБУЧЕНИЯ:

1. Указываем, сколько всего будет итераций I\_end.
2. Начальная нулевая итерация I = 0, картинка K=1:

Обучаем сеть на созданном заранее, фиксированном наборе данных setHM0.txt в течение 12-ти эпох

1. Сохраняем полученную модель сети
2. Итерация I = I + 1:

Используем метод hard-mining и области следующей картинки:

K = K + 1

1. Используем hardMiningGood для модели сети итерации I-1, получаем список хороших пар (время ожидания будет сокращено, так как используя комбинацию пар с начальной областью, для каждой точки будем перебирать 78 вариантов пар (78 преобразований), итого всего вариантов = 78 \* 400 точек = 31200
2. С помощью hardMiningGood получаем 5000 тысяч хороших «трудноотличимых» пар с наибольшим значением для модели сети итерации I-1 и сохраняем результат в файле goodHardTriples.txt.
3. Используем hardMiningBad для модели сети итерации I-1, получаем список плохих пар (здесь будем использовать 100 точек, в силу использования комбинаций пар с начальной областью, нам не нужно рассматривать все множество произвольных комбинаций 79\*79, поэтому тут кол-во пар для перебора тоже будет уменьшено:

И домножая на кол-во преобразований, получаем:

4950 \* 8 = 386100 всевозможных пар для перебора.

1. С помощью hardMiningBad получаем 5000 тысяч плохих «трудноотличимых» пар с наибольшим значением для модели сети итерации I-1 и сохраняем результат в текстовый файле badHardTriples.txt
2. С помощью программы enlargeSet.py объединяем set0.txt, badHardTriples.txt и goodHardTriples.txt, получая расширенную выборку setHM1.txt
3. Запускаем дообучение сохраненной модели итерации I-1 на выборке setHM1.txt в течение 12-ти эпох.
4. Сохраняем полученную модель сети.
5. Повторяем пункты 3-10 пока I < I\_end.

Скриншоты работы данного алгоритма показаны в Приложении А: Скриншоты работы программ.

## Результаты второго применения метода hard-mining

Согласно алгоритму, описанному в предыдущем параграфе реализовал 2 итерации. Результат представлен в таблице 12.

Таблица 12. Результат второго применения hard-mining (2 итерации)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Картинка № | | | |
| Итерация | 1 | 2 | 3 | 4 |
| 0 (lr=0.01) | 93,00% | 91,00% | 83,75% | 82,75% |
| 1 (lr=0.001) | 68,75% | 38,00% | 53,75% | 64,75% |
| 1 (lr=0.01) | 89,75% | 69,00% | 89,25% | 84,75% |

Как видно из таблицы, лучшие результаты показало начальное обучение сети. Также надо отметить, что полученная фиксированная выборка с новым способом генерирования плохих пар показывает хорошую точность. Однако, расширение выборки получением «трудноотличимых» пар по-прежнему ухудшает показатели точности модели. При этом, если мы уменьшаем скорость обучения (learning rate), то точность также падает.

## Сравнение лучшей модели сети с известными методами

Разработав эффективную (для keras) модель нейронной сети для сопоставления нам нужно сравнить наш подход с стандартными методами решения данной задачи.

В первую очередь надо обратиться к классическим алгоритмам компьютерного зрения - SIFT и SURF. Для сравнения будем использовать две картинки: исходную и повернутую на некоторый угол. Алгоритм оценки и построения сопоставлений для модели нейронной сети будет повторять уже разработанный. Таким образом, мы будем находить на оригинальной картинке ключевые точки при помощи детектора точек, затем поворачивать картинку на некоторый, получая новую. Зная угол поворота мы можем получить в точности повернутые точки. Это нам позволит после получения пар точек, сопоставленных моделью нейронной сети, узнать точно, какие точки были сопоставлены правильно, а какие – нет.

Однако, при использовании стандартных алгоритмов - SIFT и SURF мы сталкиваемся с проблемой: их дескрипторы не просто находят точки, но и присваивают им «ориентацию».

Например, для SIFT - обираются все значения градиентов и их направлений вокруг «особой» точки. Затем выбирается лучшее направление по наибольшему значению градиента.

Затем создаётся гистограмма из 36 бинов – каждый бин отвечает за 10 градусов (итого 360 градусов – всевозможные направления). Далее на основе значений и бинов (ячеек) строится гистограмма. Для исходной точки определяется направления с наибольшим значением (пиком) на гистограмме. Однако, если есть бины (как например 300-309) принимающие значение более 80% самого большого значения (пика), то создаётся еще одна «особая» точка уже с направлением этого бина.

Таким образом, для SIFT может быть создано несколько точек с одним и тем же местоположением. И соответственно, количество точек может быть неравным на двух изображениях, поэтому мы воспользуемся методом получения дескрипторов с уже готовых найденных точек.

С помощью детектора найдем координаты точек с помощью SIFT и отделим их друг от друга так, чтобы расстояние между каждыми двумя точками было не меньше 10 пикселей. Далее сгенерируем по местоположению ключевых точек первой картинки местоположение точек для второй картинки. И обернем точки в класс cv2.KeyPoint(позиция x, позиция y, радиус точки). Затем получим «плотный» (dense) дескриптор данной точки с помощью методов SIFT и SURF.

Сравним точность сопоставления точек при помощи использования SIFT, SURF – плотных дескрипторов и нашей модели сети. Проводить сравнение будем на 4-х картинках по углу наклона 20 градусов против часовой стрелки. Для сопоставления от каждой картинки берем по 400 точек из центра. Результаты представлены в таблице 13.

Таблица 13. Результаты сравнения стандартных алгоритмов с лучшей моделью сети

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Картинка №1 (площадь)  (20 град.) | Картинка №10  (пещера)  (20 град.) | Картинка №3  (ночной двор отеля)  (20 град.) | Высоко  детализированная картинка  №19  (карта Твери)  (20 градусов) |
| SIFT | 36.25% | 31.897% | 39.5% | 29% |
| SURF | 73.25% | 79.31% | 80.25% | 63.75% |
| Net6p | 93.75% | 98.851% | 96% | 85% |

Как видно из таблицы, лучшая разработанная модель сети отлично справляется с картинками разной природы, в особенности с фотографиями пещер. Наглядно сопоставление точек для таблицы 12 продемонстрировано в приложении А: Скриншоты работы программ на рисунках 28-32.

Также надо заметить, что для картинок с высокой детализированностью, как, например, спутниковые карты - модель сети показывает меньшую точность, но все равно превосходит стандартные алгоритмы компьютерного зрения.

# Заключение

В ходе работы изучена задача сопоставления ключевых точек, работа детекторов и дескрипторов стандартных алгоритмов компьютерного зрения, основы построения нейронных сетей, основные термины. Были рассмотрены различные подходы к реализации нейронной сети для решения задачи сопоставления ключевых точек изображений. Также был разработан комплекс программ для генерации обучающей выборки, обучения сети, реализации метода «hard-mining», проверки точности сети и сопоставления ключевых точек.

Основные результаты диссертационной работы:

1. Выполнен обзор литературы, посвященной проблеме построения дескрипторов с помощью нейронных сетей;
2. Разработан собственный набор вырезанных областей для создания обучающей выборки, а также оптимальный способ генерации самой обучающей выборки на этих областях для максимизации точности нейронной сети;
3. Разработана архитектура сети net6p на основе библиотеки keras для решения задачи сопоставления точек. Она представляет собой сиамскую нейронную сеть с половинками в виде сверточных сетей, критерием объединения половинок – L2-нормой и sgd-оптимизатором со скоростью обучения 0.01;
4. Получена обрезанная модель нейронной сети net6p, которая показывает наибольшую точность среди остальных моделей - 90% в среднем при сопоставлении 400 ключевых точек различных изображений.

Данная модель обучалась на выборке из 77К элементов, созданной из набора областей №4 (см. таблица 4);

1. Реализованы 2 подхода для применения метода hard-mining для повышения точности модели нейронной сети, однако в результате точность сети ухудшилась после прохода первых итераций алгоритма.
2. Проведено сравнение лучшей модели сети с известными стандартными

алгоритмами компьютерного зрения. Причем, модель сети показала

лучший результат по сравнению с дескрипторами алгоритмов SIFT и

SURF.

# Список литературы

[1] Введение // Вики-фотограмметрия [Электронный ресурс] [http://www.racurs.ru/wiki/index.php/](http://www.racurs.ru/wiki/index.php/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5)Введение (дата обращения 22.05.2018)

[2] SfM Origins: Photogrammetry and Early Vision // Department of Computer Science. Columbia University [Электронный ресурс] <http://www1.cs.columbia.edu/~jebara/htmlpapers/SFM/node4.html> (дата обращения 22.05.2018)

[3] Structure from motion // Wikipedia, The Free Encyclopedia [Электронный ресурс] <https://en.wikipedia.org/w/index.php?title=Structure_from_motion&oldid=816884383> (дата обращения 27.12.2017)

[4] Белоусов С. «LIFT: Learned Invariant Feature Transform» // Хабрахабр: интернет-сообщество индустрии высоких технологий [Электронный ресурс] <https://habrahabr.ru/post/323688/> (дата обращения 27.12.2017)

[5] Конушин А. «Сопоставление изображений и локальные особенности» // Лекции по курсу «Введение в компьютерное зрение»,

2012, стр. 45–79.

[6] ORB (Oriented FAST and Rotated BRIEF) // OpenCV-Python Tutorials [Электронный ресурс] <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb>

(дата обращения 22.05.2018)

[7] The Most Popular Language For Machine Learning Is… // IBM Community [Электронный ресурс] <https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en>

(дата обращения 22.05.2018)

[8] Rajesh Rao, Jiun-Hung Chen. Lecture 6: Features and Image Matching // Цикл лекций по Computer Vision CSE 455. Вашингтон [Электронный ресурс] <https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf>

(дата обращения 22.05.2018)

[9] Utkarsh Sinha. SIFT: Theory and Practice // AI Shack [Электронный ресурс] <http://aishack.in/tutorials/sift-scale-invariant-feature-transform-features/>

(дата обращения 22.05.2018)

[10] Bromley J., Guyon I., LeCun Y., Sickinger E., Shah R. «Signature Verification using a "Siamese" Time Delay Neural Network», 1994

[11] Павел Торгашов. Обнаружение устойчивых признаков изображения: метод SURF // Twinpeppers [Электронный ресурс] <http://twinpeppers.blogspot.ru/2012/03/surf-httphabrahabrrupost103107.html> (дата обращения 22.05.2018)

[12] MNIST (база данных) // Wikipedia, The Free Encyclopedia [Электронный ресурс] [https://ru.wikipedia.org/wiki/](https://ru.wikipedia.org/wiki/MNIST_(%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85))MNIST\_(база\_данных) (дата обращения 22.05.2018)

[13] Simo-Serra E., Trulls E., Ferraz L., Kokkinos I., Fua P., Moreno-Noguer F.:

Discriminative Learning of Deep Convolutional Feature Point Descriptors.

ICCV (2015). http://hi.cs.waseda.ac.jp/~esimo/publications/SimoSerraICCV2015.pdf

[14] Sudjit Pal. Predicting Image Similarity using Siamese Networks // Блог Salmon Run [Электронный ресурс] <http://sujitpal.blogspot.ru/2017/04/predicting-image-similarity-using.html> (дата обращения 22.05.2018)

[15] Francois C. Deep learning with Python. NY: Manning Publications Co., 2018.

[16] Примеры реализации нейронных сетей keras. Github.com. [Электронный ресурс] <https://github.com/keras-team/keras/blob/master/examples/mnist_siamese.py> (дата обращения 22.05.2018)

[17] Raia Hadsell, Sumit Chopra, Yann LeCun. Dimensionality Reduction by Learning an Invariant Mapping // Персональный сайт Яна Лекуна [Электронный ресурс] <http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pd> (дата обращения 22.05.2018)

[18] Felzenszwalb P., Girshick F., McAllester D., Ramanan D. Object detection with discriminatively trained partbased models // IEEE Transactions on Software Engineering 32(9):1627-45. 2010

[19] Multi-view Stereo Correspondence Dataset // Персональный сайт доктора Мэттью Алана Брауна [Электронный ресурс] <http://matthewalunbrown.com/patchdata/patchdata.html> (дата обращения 22.05.2018)

[20] Куча (структура данных) // Wikipedia, The Free Encyclopedia [Электронный ресурс] [https://ru.wikipedia.org/wiki/](https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D1%87%D0%B0_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85))Куча\_(структура\_данных) (дата обращения 22.05.2018)

[21] Heap queue algorithm // Python Documentation [Электронный ресурс] <https://docs.python.org/3/library/heapq.html> (дата обращения 22.05.2018)

[22] How to preserve metric values over training sessions in Keras // Stackoverflow source [Электронный ресурс] <https://stackoverflow.com/questions/46042708/how-to-preserve-metric-values-over-training-sessions-in-keras/46045360#46045360> (дата обращения 22.05.2018)

[23] Not able to resume training after loading model + weights [Электронный ресурс] <https://github.com/keras-team/keras/issues/2378> (дата обращения 22.05.2018)

[24] Deeplearning4j // Официальный сайт фреймворка [Электронный ресурс] <https://deeplearning4j.org/> (дата обращения 22.05.2018)

[25] TensorFlow // Официальный сайт фреймворка [Электронный ресурс] <https://www.tensorflow.org/> (дата обращения 22.05.2018)

[26] Theano // Официальный сайт фреймворка [Электронный ресурс] <http://deeplearning.net/software/theano/> (дата обращения 22.05.2018)

# Приложение А: Скриншоты работы программ

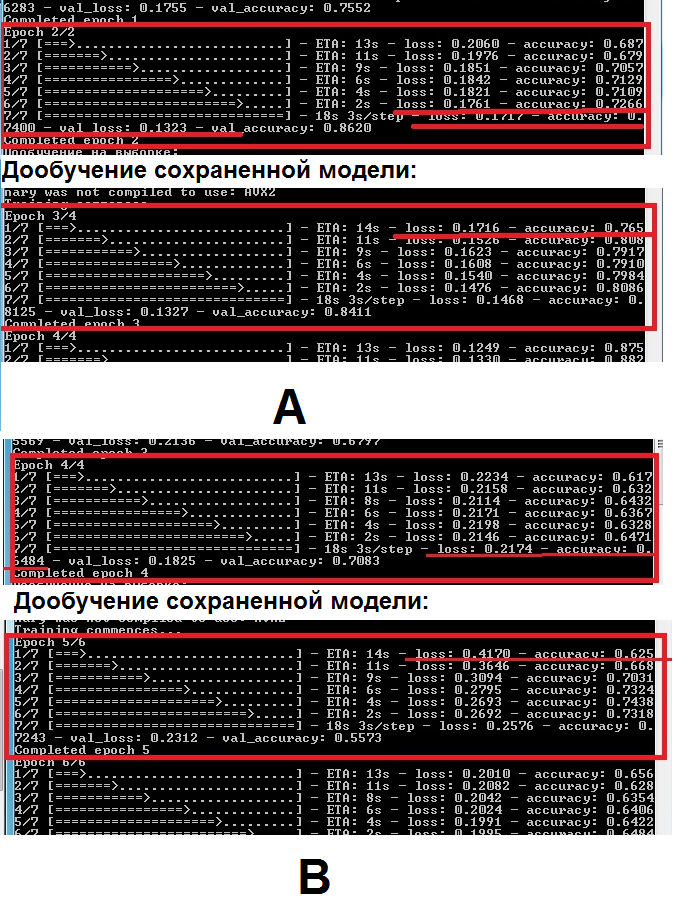


Рисунок 25. Дообучение моделей: А – выборка не меняется, В – меняется

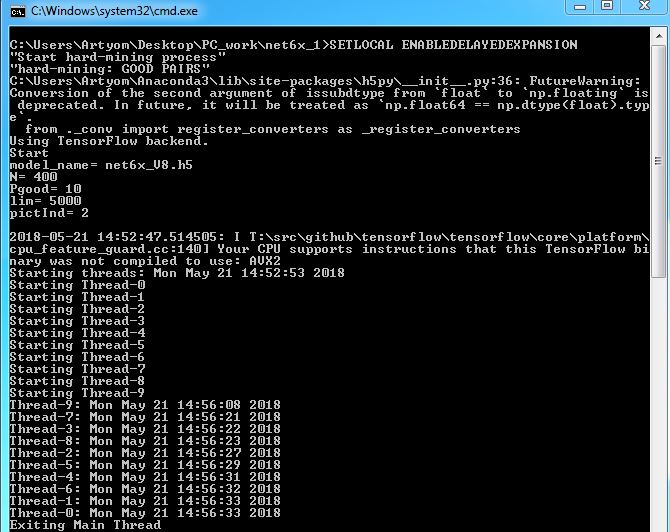


Рисунок 26. Работа метода "hard-mining" для поиска хороших пар

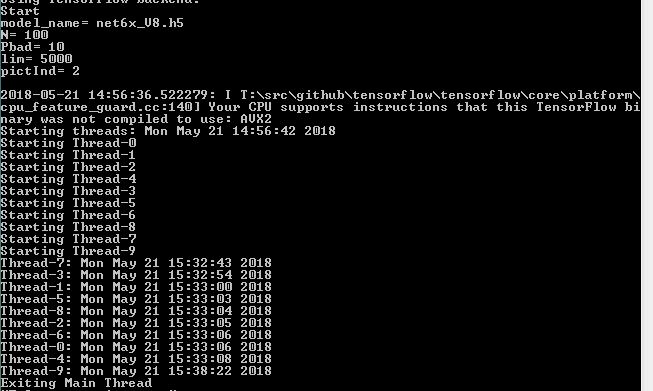


Рисунок 27. Работа метода "hard-mining" для поиска плохих пар

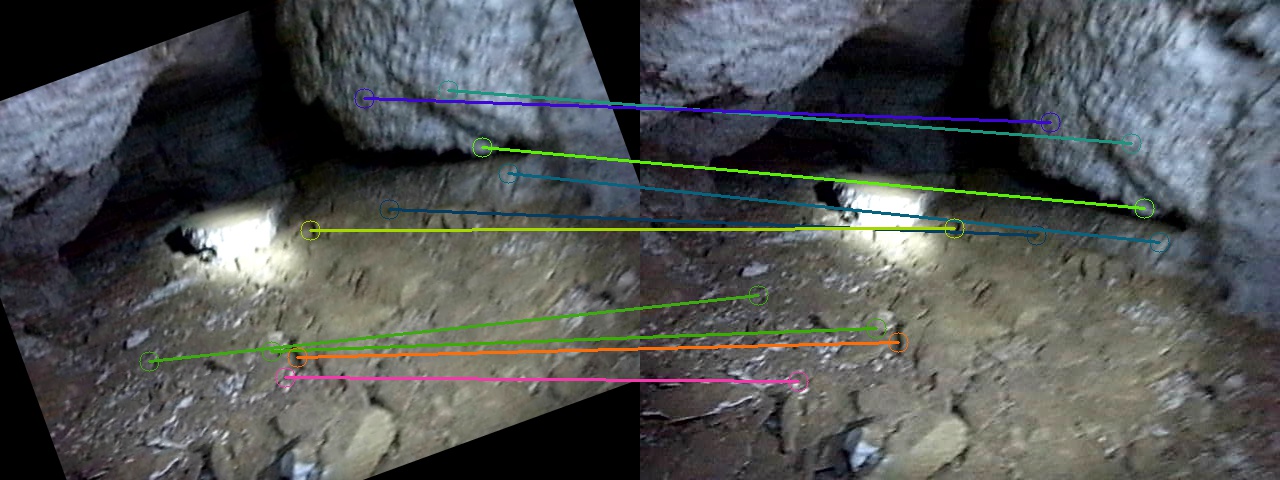


Рисунок 28. Сопоставление точек при помощи модели сети Net6p (картинка №3 – пещера)



Рисунок 29. Сопоставление точек при помощи модели сети Net6p (картинка №10 – ночной двор отеля)

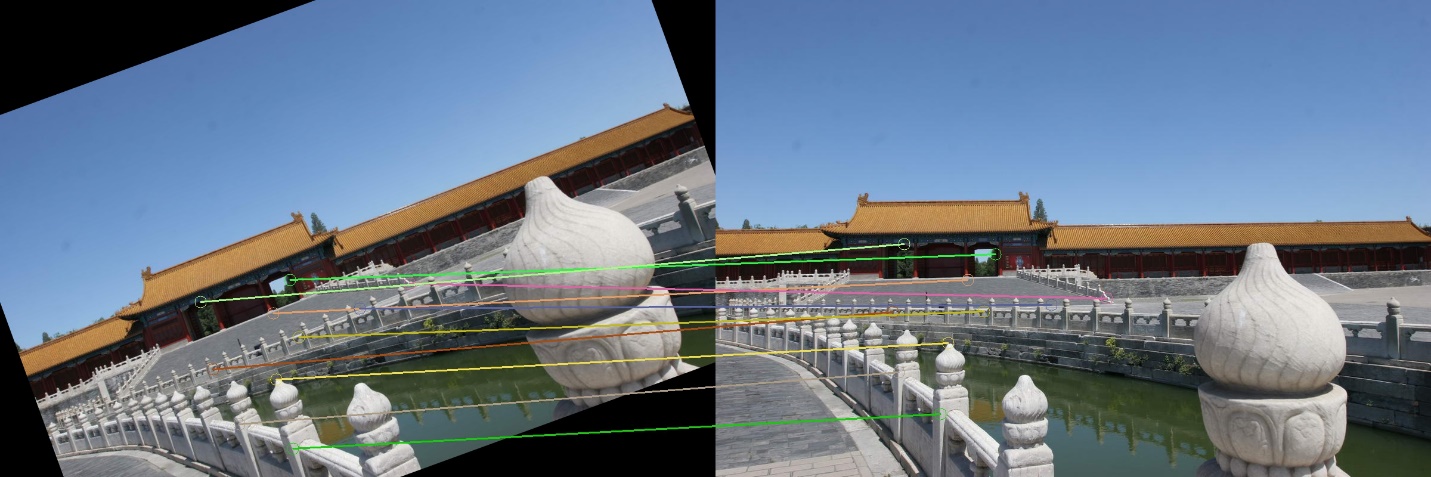


Рисунок 30. Сопоставление точек при помощи модели сети Net6p (картинка №1 – площадь)



Рисунок 31. Сопоставление точек при помощи модели сети Net6p (картинка №19 – карта Твери)

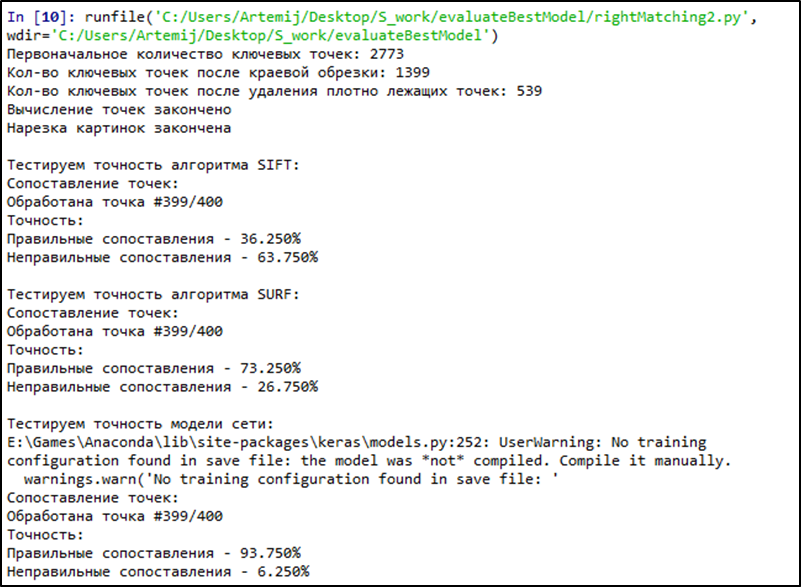


Рисунок 32. Скриншот работы программы сравнения сопоставлений точек разными методами

# Приложение Б: Исходные коды программ

В данном приложении приводится код основных программ, использованных для обучения, проверки сети и сопоставления точек.

## Create\_training\_set3.py

Программа для генерации набора вырезанных областей №4

import cv2

import numpy as np

import random

from random import shuffle

import os

import matplotlib.pyplot as plt

import itertools

from scipy.misc import imresize

#поворот координат:

def rotateXYaroundCenter(coords, fi, center):

    x, y = coords

    M = np.array(cv2.getRotationMatrix2D((center[0], center[1]), fi, 1))

    old\_coords = np.array([x, y, 1])

    new\_coords = M.dot(old\_coords)

    return new\_coords[0], new\_coords[1]

#поворот изображения:

def rotateImage(img, fi):

    num\_rows, num\_cols = img.shape[:2]

    rotation\_matrix = cv2.getRotationMatrix2D((num\_cols/2, num\_rows/2), fi, 1)

    img\_rotation = cv2.warpAffine(img, rotation\_matrix, (num\_cols, num\_rows))

    return img\_rotation

#изменить яркость изображения:

def increase\_brightness(img, value=30):

    #перевод изображения из RedGreenBlue в формат HueSaturationValue (тон, насыщенность, значение)

    hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

    h, s, v = cv2.split(hsv)

    lim = 255 - value

    #для исключения случая overflow (пиксель может принимать только значения из [0...255])

    v[v > lim] = 255

    v[v <= lim] += value

    final\_hsv = cv2.merge((h, s, v))

    img = cv2.cvtColor(final\_hsv, cv2.COLOR\_HSV2BGR)

    return img

def noisy(noise\_typ,image):

    if noise\_typ == "gauss":

        row,col,ch= image.shape

        mean = 0

        var = 0.1

        sigma = var\*\*0.5

        gauss = np.random.normal(mean,sigma,(row,col,ch))

        gauss = gauss.reshape(row,col,ch)

        noisy = image + gauss

        return noisy

    elif noise\_typ == "s&p":

        row,col,ch = image.shape

        s\_vs\_p = 0.5

        amount = 0.004

        out = np.copy(image)

        # Salt mode

        num\_salt = np.ceil(amount \* image.size \* s\_vs\_p)

        coords = [np.random.randint(0, i - 1, int(num\_salt))

              for i in image.shape]

        out[coords] = 1

        # Pepper mode

        num\_pepper = np.ceil(amount\* image.size \* (1. - s\_vs\_p))

        coords = [np.random.randint(0, i - 1, int(num\_pepper))

              for i in image.shape]

        out[coords] = 0

        return out

    elif noise\_typ == "poisson":

        vals = len(np.unique(image))

        vals = 2 \*\* np.ceil(np.log2(vals))

        noisy = np.random.poisson(image \* vals) / float(vals)

        return noisy

    elif noise\_typ =="speckle":

        row,col,ch = image.shape

        gauss = np.random.randn(row,col,ch)

        gauss = gauss.reshape(row,col,ch)

        noisy = image + image \* gauss

        return noisy

#сгенерировать обучающую выборку:

def generateTrainingSet(pathToRead, pathToSave, imageName, limit=400):

    #прочитаем изображение

    imgA = cv2.imread(pathToRead)

    #номер изображения:

    imgClass = imageName.split('.')[0]

    '''Найдем ключевые точки в центральной области'''

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

    #размер области для рассматриваемой ключевой точки: 20x20:

    areaSize = 50

    N = areaSize//2

    #размер области для взятия особых точек

    #возьмём только точки около центра в радиусе r = (h - areaSize)/2, где h - наименьший параметр размера изображения

    h = imgA.shape[0]

    if imgA.shape[0] >= imgA.shape[1]:

        h = imgA.shape[1]

    r = (h - areaSize)/2

    #посчитаем центр изображения:

    imgCenter = [imgA.shape[1]/2, imgA.shape[0]/2,]

    #детектор:

    detector = cv2.xfeatures2d.SIFT\_create()

    #находим особые точки:

    kps = detector.detect(gray, None)

    #берём особые точки рядом с центром из множества детектора sift:

    knew = [kps[i] for i in range(len(kps)) if (kps[i].pt[0] - imgCenter[0])\*\*2 + (kps[i].pt[1] - imgCenter[1])\*\*2 <= r\*\*2 ]

    '''Выполняем преобразования точек'''

    '''Простые:'''

    trNum = 0 #номер преобразования

    #ограничим число точек, попадающих в набор:

    number = limit if len(knew) >= limit else len(knew)

    #цикл по всем точкам:

    for i in range(number):

        #вырезаем область точки (но из ЧБ изображения):

        crop\_imgA = imgA[(int)(knew[i].pt[1]-N):(int)(knew[i].pt[1]+N), (int)(knew[i].pt[0]-N):(int)(knew[i].pt[0]+N)]

        #заготока наименования области:

        img\_name = str(i) #номер области

        img\_name = img\_name + '0'\*(3 - len(imgClass)) + str(imgClass) #номер кадра (класса изображения)

        '''Нулевое преобразование:'''

        trNum = 0 #номер преобразования

        #создаём имя облатси:

        img\_name0 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

        #сохраняем область:

        cv2.imwrite(os.path.join(pathToSave, img\_name0), cv2.cvtColor(crop\_imgA, cv2.COLOR\_BGR2GRAY))

        '''Шум:'''

        trNum += 1

        #преобразование

        noiseImg = noisy("s&p", crop\_imgA)

        noiseImg = cv2.cvtColor(noiseImg, cv2.COLOR\_BGR2GRAY)

        #создаём имя облатси:

        img\_name0 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

        #сохраняем область:

        cv2.imwrite(os.path.join(pathToSave, img\_name0), noiseImg)

        '''Масштаб:'''

        trNum += 1

        #преобразование x1

        scaleImg1 = cv2.GaussianBlur(crop\_imgA, (3,3), 0)

        grayScaleImg1 = cv2.cvtColor(scaleImg1, cv2.COLOR\_BGR2GRAY)

        #создаём имя облатси:

        img\_name0 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

        #сохраняем область:

        cv2.imwrite(os.path.join(pathToSave, img\_name0), grayScaleImg1)

        trNum += 1

        #преобразование x2

        scaleImg2 = cv2.GaussianBlur(scaleImg1, (3,3), 0)

        grayScaleImg2 = cv2.cvtColor(scaleImg2, cv2.COLOR\_BGR2GRAY)

        #создаём имя облатси:

        img\_name1 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

        #сохраняем область:

        cv2.imwrite(os.path.join(pathToSave, img\_name1), grayScaleImg2)

        '''Яркость:'''

        trNum += 1

        values = [20, 40, 60, 80]

        for val in values:

            brightImg = increase\_brightness(crop\_imgA, value=val)

            brightImg = cv2.cvtColor(brightImg, cv2.COLOR\_BGR2GRAY)

            #создаём имя облатси:

            img\_name0 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

            #сохраняем область:

            cv2.imwrite(os.path.join(pathToSave, img\_name0), brightImg)

            trNum += 1 #следуюший номер преобразования

        print("Простые преобразования: обработана точка #", i)

    print("Простые преобразования: завершено")

    '''Сложные:'''

    '''Поворот:'''

    #цикл по углам поворота:

    for fi in range(5, 360, 5):

        #поворачиваем изображение на угол:

        rotatedImg = rotateImage(gray, fi)

        #идем по точкам:

        for i in range(number):

            #находим соответствующую точку на повернутом изображении:

            #вращением вокруг центра изображения:

            coords = knew[i].pt[0], knew[i].pt[1]

            xr, yr = rotateXYaroundCenter(coords, fi, imgCenter)

            #вырезаем область:

            crop\_rotatedImg = rotatedImg[(int)(yr-N):(int)(yr+N), (int)(xr-N):(int)(xr+N)]

            #создаём имя облатси:

            #заготока наименования области:

            img\_name = str(i) #номер области

            img\_name = img\_name + '0'\*(3 - len(imgClass)) + str(imgClass) #номер кадра (класса изображения)

            img\_name0 = img\_name + '0'\*(3 - len(str(trNum))) + str(trNum) + '.jpg'

            #сохраняем область:

            cv2.imwrite(os.path.join(pathToSave, img\_name0), crop\_rotatedImg)

        print("Сложные преобразования: выполнено для угла ", fi)

        #следуюший номер преобразования

        trNum += 1

    print("Сложные преобразования: завершено")

def countFilesInDest(directory):

    s = 0

    for img\_file in os.listdir(directory):

        s += 1

    return s

if \_\_name\_\_ == '\_\_main\_\_':

    DATA\_DIR = os.getcwd() #текущая директория

    READ\_DIR = os.path.join(DATA\_DIR, "source")

    SAVE\_DIR = os.path.join(DATA\_DIR, "produced\_data3")

    for img\_file in os.listdir(READ\_DIR):

        print("Изображение: ", img\_file)

        generateTrainingSet(os.path.join(READ\_DIR, img\_file), SAVE\_DIR, img\_file, 400)

    print(countFilesInDest(SAVE\_DIR))

## Do\_data\_analysis.py

Первая программа для сопоставления точек и проверки точности

модели сети. Do\_data\_analysis.py:

import cv2

import numpy as np

import random

from random import shuffle

import os

import matplotlib.pyplot as plt

import itertools

from scipy.misc import imresize

from scipy.spatial import distance

from keras.models import load\_model

from keras.models import Model # основной класс для настройки и обучения нейронной сети

from keras.layers import Input, Dense # два типа слоев нейронной сети, которые мы будем использовать

from keras.utils import np\_utils

from keras import backend as K

import pandas as pd

#поворот изображения:

def rotateImg(filename, fi):

    img = cv2.imread(filename)

    num\_rows, num\_cols = img.shape[:2]

    rotation\_matrix = cv2.getRotationMatrix2D((num\_cols/2, num\_rows/2), fi, 1)

    img\_rotation = cv2.warpAffine(img, rotation\_matrix, (num\_cols, num\_rows))

    #cv2.imwrite('rotImg.jpg', img\_rotation)

    return img\_rotation

#поворот координат:

def rotateXYaroundCenter(coords, fi, center):

    x, y = coords

    M = np.array(cv2.getRotationMatrix2D((center[0], center[1]), fi, 1))

    old\_coords = np.array([x, y, 1])

    new\_coords = M.dot(old\_coords)

    return new\_coords[0], new\_coords[1]

#нарисовать точки (skp и tkp - обычные списки: [[x,y], ...])

def drawKeyPoints(img, template, skp, tkp, num=-1):

    h1, w1 = img.shape[:2]

    h2, w2 = template.shape[:2]

    nWidth = w1+w2

    nHeight = max(h1, h2)

    hdif = (h1-h2)//2

    newimg = np.zeros((nHeight, nWidth, 3), np.uint8)

    newimg[hdif:hdif+h2, :w2] = template

    newimg[:h1, w2:w1+w2] = img

    maxlen = min(len(skp), len(tkp))

    if num < 0 or num > maxlen:

        num = maxlen

    #отрисовка:

    radius = 10

    for i in range(num):

        pt\_a = (int(tkp[i][0]), int(tkp[i][1]+hdif))

        pt\_b = (int(skp[i][0]+w2), int(skp[i][1]))

        newColor = np.random.randint(0,256,3)

        newColor = (int(newColor[0]), int(newColor[1]), int(newColor[2]))

        cv2.circle(newimg, pt\_a, radius, newColor)

        cv2.circle(newimg, pt\_b, radius, newColor)

        cv2.line(newimg, pt\_a, pt\_b, (newColor), 2)

    return newimg

#показать изображение:

def show\_img(sid, img\_file, img\_title):

    plt.subplot(sid)

    plt.title(img\_title)

    plt.xticks([])

    plt.yticks([])

    plt.imshow(img\_file)

#убрать точки так, чтобы минимальное расстояние между оставшимися было dist:

#для вычисления расстояния используется евклидова метрика:

def more\_space\_for(a, dist):

    i = 0

    while i < len(a):

        j = i+1

        while j < len(a):

            if distance.euclidean(a[i], a[j]) < dist:

                a.pop(j)

            else:

                j += 1

        i += 1

    return a

'''Получить точки для анализа точности способ 1:

       img1, img2 - две картинки для поиска точек

       для второй картинки точки получаются поворотом точек из первой

       fi - угол поворота (в градусах)

       numP - количество точек для сопоставления

       min\_dist - минимальное расстояние между найденными точками

'''

def getPointsFromImagesW1(img1, img2, fi, numP, min\_dist):

    # Initiate SIFT detector

    det = cv2.xfeatures2d.SIFT\_create()

    gray1= cv2.cvtColor(img1,cv2.COLOR\_BGR2GRAY)

    '''вычислим расположение точек на картинке 1 (только координаты)'''

    kp1 = det.detect(gray1,None)

    print('Первоначальное количество ключевых точек:', len(kp1))

    '''Подрезка точек: возьмём центральные:'''

    areaSize = 20

    #размер области для взятия особых точек

    #возьмём только точки около центра в радиусе r = (h - areaSize)/2, где h - наименьший параметр размера изображения

    h = img1.shape[0]

    if img1.shape[0] >= img1.shape[1]:

        h = img1.shape[1]

    r = (h - areaSize)/2

    #посчитаем центр изображения:

    imgCenter = [img1.shape[1]/2, img1.shape[0]/2,]

    #берём особые точки рядом с центром из множества детектора sift:

    knew1 = [[kp1[i].pt[0], kp1[i].pt[1]] for i in range(len(kp1)) if (kp1[i].pt[0] - imgCenter[0])\*\*2 + (kp1[i].pt[1] - imgCenter[1])\*\*2 <= r\*\*2 ]

    print('Кол-во ключевых точек после краевой обрезки:', len(knew1))

    '''Подрезка точек: оставляем только те, которые находятся

      по краней мере на min\_dist расстоянии от других

   '''

    knew1 = more\_space\_for(knew1, min\_dist)

    print('Кол-во ключевых точек после удаления плотно лежащих точек:', len(knew1))

#    radius = 10

#    for p in knew1:

#        cv2.circle(img1, ((int)(p[0]),(int)(p[1])), radius, (255, 0, 0))

#

#    cv2.imshow('keypoints.jpg',img1)

#    cv2.waitKey(0)

#    cv2.destroyAllWindows()

#

    '''вычислим расположение точек на картинке 2, поворотом'''

    knew2 = knew1.copy()

    for i in range(len(knew2)):

        coords = knew2[i][0], knew2[i][1]

        n\_coords = rotateXYaroundCenter(coords, fi, imgCenter)

        knew2[i] = n\_coords

    print('Вычисление точек закончено')

    '''Вырежем изображения из картинок 1 и 2:'''

    N = areaSize//2

    mas1 = []

    mas2 = []

    #показать следующее количество точек:

    numPoints = numP if numP <= len(knew1) else len(knew1)

    for i in range(numPoints):

        #вырезаем область:

        mas1.append([img1[(int)(knew1[i][1]-N):(int)(knew1[i][1]+N), (int)(knew1[i][0]-N):(int)(knew1[i][0]+N)], i])

        mas2.append([img2[(int)(knew2[i][1]-N):(int)(knew2[i][1]+N), (int)(knew2[i][0]-N):(int)(knew2[i][0]+N)], i])

    print('Нарезка картинок закончена')

#    show\_img(131, mas1[9][0], "original")

#    show\_img(132, mas2[9][0], "similar")

#    plt.tight\_layout()

#    plt.show()

    '''Для чистоты проверки, перемешаем массивы:'''

    mas1 = np.random.permutation(mas1)

    mas2 = np.random.permutation(mas2)

    return knew1, knew2, mas1, mas2

'''АНАЛИЗ ТОЧНОСТИ МОДЕЛИ для СЕТИ 0: Способ 1'''

def analyze\_modelW0(knew1, knew2, mas1, mas2, modelName):

    '''Загрузим натренированную модель:'''

    model = load\_model(modelName)

    '''сопоставляем точки картинок:'''

    right = 0

    wrong = 0

    #массивы точек для отрисовки:

    skp = []

    tkp = []

    #сделаем так, чтобы точки из массивов удалялись:

    i = 0

    while len(mas1) > 0:

        #берём область из массива точек картинки 1

        #настроим формат картинки:

        obj1 = np.array(mas1[0][0]).astype(np.float32)

        obj1 = obj1.reshape(1, 20, 20, 3)

        obj1 = np.divide(obj1, 256) # Normalise data to [0, 1] range

        #и ищем наиболее подходящий вариант во втором массиве:

        all\_pred = []

        for j in range(len(mas2)):

            #настроим формат картинки:

            obj2 = np.array(mas2[j][0]).astype(np.float32)

            obj2 = obj2.reshape(1, 20, 20, 3)

            obj2 = np.divide(obj2, 256) # Normalise data to [0, 1] range

            #получаем предсказание:

            prediction = model.predict([np.array(obj1), np.array(obj2)])

            #складываем вероятности того, что

            #текущей области из массива mas1 соответствует область mas2[j]

            all\_pred.append(prediction[0][1])

        #выбираем наиболее подходящее предсказание:

        #наибольшая вероятность сходства двух областей

        ind = np.argmax(all\_pred)

        #проверяем описывают ли картинки одну и ту же область:

        if mas1[0][1] == mas2[ind][1]:

            right += 1

        else:

            wrong += 1

        skp.append(knew1[mas1[0][1]])

        tkp.append(knew2[mas2[ind][1]])

        print('Обработана точка #', i)

        i += 1

        # удаляем точки из массивов mas1 и mas2:

        mas1 = np.delete(mas1, 0, 0)

        mas2 = np.delete(mas2, ind, 0)

    print('Точность сопоставления точек нейронной сетью:')

    positive\_acc = (right/(right+wrong))\*100

    negative\_acc = (wrong/(right+wrong))\*100

    print('Правильные сопоставления - {0:.3f}%'.format(positive\_acc))

    print('Неправильные сопоставления - {0:.3f}%'.format(negative\_acc))

    return skp, tkp, positive\_acc/100

def contrastive\_loss(y\_true, y\_pred):

    '''Contrastive loss from Hadsell-et-al.'06

   http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

   '''

    margin = 1

    return K.mean(y\_true \* K.square(y\_pred) +

                  (1 - y\_true) \* K.square(K.maximum(margin - y\_pred, 0)))

'''АНАЛИЗ ТОЧНОСТИ МОДЕЛИ для СЕТИ 1: Способ 1'''

def analyze\_modelW1(knew1, knew2, mas1, mas2, modelName):

    '''Загрузим натренированную модель:'''

    model = load\_model(modelName, custom\_objects={'contrastive\_loss': contrastive\_loss})

    '''сопоставляем точки картинок:'''

    right = 0

    wrong = 0

    #массивы точек для отрисовки:

    skp = []

    tkp = []

    #сделаем так, чтобы точки из массивов удалялись:

    i = 0

    while len(mas1) > 0:

        #берём область из массива точек картинки 1

        #настроим формат картинки:

        obj1 = np.array(mas1[0][0]).astype(np.float32)

        obj1 = obj1.reshape(1, 20, 20, 3)

        obj1 = np.divide(obj1, 255) # Normalise data to [0, 1] range

        #и ищем наиболее подходящий вариант во втором массиве:

        all\_pred = []

        for j in range(len(mas2)):

            #настроим формат картинки:

            obj2 = np.array(mas2[j][0]).astype(np.float32)

            obj2 = obj2.reshape(1, 20, 20, 3)

            obj2 = np.divide(obj2, 255) # Normalise data to [0, 1] range

            #получаем предсказание:

            prediction = model.predict([np.array(obj1), np.array(obj2)])

            #складываем вероятности того, что

            #текущей области из массива mas1 соответствует область mas2[j]

            all\_pred.append(prediction[0][0])

        #выбираем наиболее подходящее предсказание:

        #наибольшая вероятность сходства двух областей

        ind = np.argmin(all\_pred)

        #проверяем описывают ли картинки одну и ту же область:

        if mas1[0][1] == mas2[ind][1]:

            right += 1

        else:

            wrong += 1

        skp.append(knew1[mas1[0][1]])

        tkp.append(knew2[mas2[ind][1]])

        print('Обработана точка #', i)

        i += 1

        # удаляем точки из массивов mas1 и mas2:

        mas1 = np.delete(mas1, 0, 0)

        mas2 = np.delete(mas2, ind, 0)

    print('Точность сопоставления точек нейронной сетью:')

    positive\_acc = (right/(right+wrong))\*100

    negative\_acc = (wrong/(right+wrong))\*100

    print('Правильные сопоставления - {0:.3f}%'.format(positive\_acc))

    print('Неправильные сопоставления - {0:.3f}%'.format(negative\_acc))

    return skp, tkp, positive\_acc/100

#визуальная проверка точек:

def checkTurningPointsVisually():

    img1 = cv2.imread('a\_a1.jpg')

    img2 = rotateImg('a\_a1.jpg', 45) #поворот второй картинки

    knew1, knew2, mas1, mas2 = getPointsFromImagesW1(img1,\

                                                     img2, \

                                                     45, \

                                                     30, \

                                                     10)

    i = 0

    j = 0

    for i in range(len(mas1)):

        for j in range(len(mas2)):

            if mas1[i][1] == mas2[j][1]:

                break

    show\_img(131, mas1[i][0], "original")

    show\_img(132, mas2[j][0], "similar")

    plt.tight\_layout()

    plt.show()

'''ОСНОВНАЯ ПРОГРАММА ПРОВЕРКИ:'''

def main():

    '''

   #запускаем тест искуственного анализа:

   #он находит точки на другом изображении простым поворотом на нек. угол fi

   #уже найденных детектором sift точек на первом изображении:

   проверяем углы: 50, 100

   количество сопоставлемых точек: 500

   '''

    angles = [50]

    numMatchingPoints = 400

    min\_distance\_between\_points = 10

    pictures\_for\_test = ['a\_a4.jpg']

    base\_name = 'mySiameseArtificial\_model'

    base\_name1 = 'mySiameseNet1\_model'

    models\_for\_test = [base\_name+'\_V1.h5', \

                       base\_name+'.h5', \

                       base\_name+'\_V2.h5', \

                       base\_name+'\_V3.h5', \

                       base\_name1+'\_V0.h5', \

                       base\_name1+'\_V1.h5']

    #сохранить результаты: в датафрейм:

    indexes=[base\_name+'\_V1+\n(900000)', \

             base\_name+'+\n(239054)', \

             base\_name+'\_V2+\n(200000)', \

             base\_name+'\_V3+\n(100000)', \

             base\_name1+'\_V0.h5(100000)', \

             base\_name1+'\_V1.h5(239054)']

    data = pd.DataFrame(columns=['Accuracy'])

    #тестирование:

    for picture in pictures\_for\_test:

        print('Картинка: ', picture)

        for angle in angles:

            print('Угол поворота:', angle)

            '''Инициализация картинок'''

            img1 = cv2.imread(picture)

            img2 = rotateImg(picture, angle) #поворот второй картинки

            '''Получение ключевых точек 1 способом:'''

            knew1, knew2, mas1, mas2 = getPointsFromImagesW1(img1,\

                                                             img2, \

                                                             angle, \

                                                             numMatchingPoints, \

                                                             min\_distance\_between\_points)

            '''Анализ моделей:'''

            ind = 0

            for model in models\_for\_test:

                rez\_row = []

                print('Тестируем модель ', model)

                skp, tkp, accuracy = [], [], 0

                '''Для сети №1'''

                if model.find(base\_name1) != -1:

                    skp, tkp, accuracy = analyze\_modelW1(knew1, knew2, mas1, mas2, model)

                else:

                    '''ДЛя сети №0'''

                    skp, tkp, accuracy = analyze\_modelW0(knew1, knew2, mas1, mas2, model)

                #добавляем точность модели:

                rez\_row.append(accuracy)

                '''отрисовка точек: '''

#                imgRez = drawKeyPoints(img1, img2, skp[:50], tkp[:50])

#                cv2.imwrite('MKP\_{0}\_{1}\_angle\_{2}\_.jpg'.format(picture.replace('.jpg', ''), model.replace('.h5', ''), angle),imgRez)

#

                '''Сохранение данных анализа:'''

                new\_row = pd.DataFrame([rez\_row], \

                                       index=[indexes[ind]], \

                                       columns=['Accuracy'])

                data = data.append(new\_row)

                ind += 1

    print(data)

    #сохранить всё:

    writer = pd.ExcelWriter('testResults1.xlsx')

    data.to\_excel(writer,'Sheet1')

    writer.save()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

## net6p.py

Программа для обучения лучшей модели сети на основе архитектуры net6p

import os

import json

import matplotlib.pyplot as plt

import numpy as np

from keras.models import Sequential, Model

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.layers.core import Activation

from keras.layers import Input, Flatten, Dense, Dropout, Lambda, BatchNormalization

from keras.optimizers import RMSprop, SGD

from keras import backend as K

from keras.models import load\_model

from keras.utils import np\_utils

from keras import callbacks

from random import shuffle

import time

import cv2

from sys import stdout

import re

import sys

''' ОПИСАНИЕ КЛАССА ФИКСАТОРА СОСТОЯНИЙ СЕТИ ВО ВРЕМЯ ОБУЧЕНИЯ'''

# State monitor callback. Tracks how well we are doing and writes

# some state to a json file. This lets us resume training seamlessly.

#

# ModelState.state is:

#

# { "epoch\_count": nnnn,

#   "best\_values": { dictionary with keys for each log value },

#   "best\_epoch": { dictionary with keys for each log value }

# }

class ModelState(callbacks.Callback):

    def \_\_init\_\_(self, state\_path):

        self.state\_path = state\_path

        if os.path.isfile(state\_path):

            print('Loading existing .json state')

            with open(state\_path, 'r') as f:

                self.state = json.load(f)

        else:

            self.state = { 'epoch\_count': 0,

                           'best\_values': {},

                           'best\_epoch': {}

                         }

    def on\_train\_begin(self, logs={}):

        print('Training commences...')

    def on\_epoch\_end(self, batch, logs={}):

        # Currently, for everything we track, lower is better

        for k in logs:

            if k not in self.state['best\_values']:

                self.state['best\_values'][k] = float(logs[k])

                self.state['best\_epoch'][k] = self.state['epoch\_count']

            else:

                if ('accuracy' not in k) and logs[k] < self.state['best\_values'][k]:

                    self.state['best\_values'][k] = float(logs[k])

                elif ('accuracy' in k) and logs[k] > self.state['best\_values'][k]:

                    self.state['best\_values'][k] = float(logs[k])

                    self.state['best\_epoch'][k] = self.state['epoch\_count']

        self.state['epoch\_count'] += 1

        print('Completed epoch', self.state['epoch\_count'])

        with open(self.state\_path, 'w') as f:

            json.dump(self.state, f, indent=4)

'''ОПИСАНИЕ МОДЕЛИ СЕТИ:'''

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

def euclidean\_distance(vects):

    x, y = vects

    return K.sqrt(K.maximum(K.sum(K.square(x - y), axis=1, keepdims=True), K.epsilon()))

def eucl\_dist\_output\_shape(shapes):

    shape1, shape2 = shapes

    return (shape1[0], 1)

def contrastive\_loss(y\_true, y\_pred):

    '''Contrastive loss from Hadsell-et-al.'06

   http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

   '''

    margin = 1

    return K.mean(y\_true \* K.square(y\_pred) +

                  (1 - y\_true) \* K.square(K.maximum(margin - y\_pred, 0)))

#убрали нормализацию:

#tanh

def create\_cnn\_base\_network(input\_shap):

    seq = Sequential()

    seq.add(Conv2D(32, kernel\_size=7, padding="same", input\_shape=input\_shap))

    seq.add(Activation("tanh"))

    seq.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

    seq.add(Conv2D(64, kernel\_size=6, padding="same"))

    seq.add(Activation("tanh"))

    seq.add(MaxPooling2D(pool\_size=(3, 3), strides=(3, 3)))

    seq.add(Conv2D(128, kernel\_size=5, padding="same"))

    seq.add(Activation("tanh"))

    seq.add(MaxPooling2D(pool\_size=(4, 4), strides=(4, 4)))

    seq.add(Flatten())

    seq.add(Dense(128))

    return seq

def compute\_accuracy(y\_true, y\_pred):

    '''Compute classification accuracy with a fixed threshold on distances.

   '''

    pred = y\_pred.ravel() < 0.5

    return np.mean(pred == y\_true)

def accuracy(y\_true, y\_pred):

    '''Compute classification accuracy with a fixed threshold on distances.

   '''

    return K.mean(K.equal(y\_true, K.cast(y\_pred < 0.5, y\_true.dtype)))

'''Подготовим обучающую выборку:'''

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

#взять произвольное изображение

def get\_random\_image(img\_groups, group\_names, gid):

    gname = group\_names[gid]

    photos = img\_groups[gname]

    pid = np.random.choice(np.arange(len(photos)), size=1)[0]

    pname = photos[pid]

    return gname + pname + ".jpg"

#загрузка изображений:

image\_cache = {}

def load\_image(image\_name):

    if not image\_cache.\_\_contains\_\_(image\_name):

        image = cv2.imread(os.path.join(IMAGE\_DIR, image\_name), cv2.IMREAD\_GRAYSCALE).astype(np.float32)

        #НОРМИРОВКА: [0:255] -> [0:1]:

        image = np.divide(image, 255)

        image\_cache[image\_name] = np.reshape(image, (50, 50, 1))

    return image\_cache[image\_name]

'''Генерации данных:'''

def create\_triples(rezSize, images):

    ''' Функция перебором создаёт тройки для валидации после обучения

       rezSize - параметр устанавливает, сколько данных нужно в итоге получить

       images - массив номеров изображений, которые будем использовать для валидации

       Возвращает список данных для валидации сети.

   '''

    #распределяем изображения по группам (отдельная группа для отдельной области)

    img\_groups = {}

    print('Произвольная генерация данных:')

    i\_lim = 0

    limit = rezSize//2

    '''сделаем постепенную загрузку картинок (ведь, мы знаем сколько их)'''

    #для одной картинки вырезано 400 точек по 79 преобразований на каждую:

    #нумерация точек: от 0 до 399(pointsPerPict)

    #нумерация картинок: от 1 до 16(imageCount)

    #нумерация преобразований: от 0 до 78

    pointsPerPict = 400

    trans = 79

    #цикл по картинкам:

    for i in images:

        curr\_pict = '0'\*(3 - len(str(i))) + str(i)

        #цикл по точкам:

        for j in range(0, pointsPerPict, 1):

            #группа:

            aid = str(j) + curr\_pict

            img\_groups[aid] = []

            #цикл по реобразованиям точек:

            for k in range(0, trans):

                img\_groups[aid].append('0'\*(3 - len(str(k))) + str(k))

                i\_lim += 1

                if i\_lim >= limit:

                    break

            if i\_lim >= limit:

                    break

        if i\_lim >= limit:

            break

    #создаем хорошие и плохие тройки (изображения одной области и разных соотв.)

    pos\_triples, neg\_triples = [], []

    #позитивные пары - это комбинация изображений из одной группы

    print('Хорошие тройки:')

    for key in img\_groups.keys():

        print('Область %s' % (key))

        triples = []

        for i in range(len(img\_groups[key])):

            #triples.append((key + '0'\*(3 - len(str(i))) + str(i) + ".jpg", key + "000" + ".jpg", 1))

            triples.append((key + img\_groups[key][i] + '.jpg', key + '000' + '.jpg', 1))

            stdout.write("\rтройка # %d / %d" % (i, len(img\_groups[key])-1))

            stdout.flush()

        pos\_triples.extend(triples)

        print("")

    #нужно такое же число плохих:

    print('Плохие тройки:')

    group\_names = list(img\_groups.keys())

    for i in range(len(pos\_triples)):

        #подбор плохих пар только из точек одного изображения

        #делается для того, чтобы сеть лучше различала похожие точки

        g1, g2 = '',''

        while True:

            g1, g2 = np.random.choice(np.arange(len(group\_names)), size=2, replace=False)

            if group\_names[g1][-3:] == group\_names[g2][-3:] and \

               group\_names[g1][:-3] != group\_names[g2][:-3]:

                break

        left = get\_random\_image(img\_groups, group\_names, g1)

        right = get\_random\_image(img\_groups, group\_names, g2)

        neg\_triples.append((left, right, 0))

        stdout.write("\rтройка # %d / %d" % (i, len(pos\_triples)-1))

        stdout.flush()

    print("")

    #добавляем плохие к хорошим:

    pos\_triples.extend(neg\_triples)

    #перемешиваем:

    shuffle(pos\_triples)

    return pos\_triples

def get\_triples\_from\_file(goodTripFile, badTripFile):

    ''' Функция считывает тройки с наименованиями областей из файлов.

       Возвращает список данных для обучения сети.

   '''

    print('Генерация данных (hard-mining):')

    triples = []

    #читаем информацию из файлов:

    #хорошие тройки:

    with open(goodTripFile, 'r') as text\_file:

        for line in text\_file:

            line = line.replace('\n', '')

            splited = line.split(' ')

            triples.append((splited[1], splited[2], int(splited[3])))

    print('Считали хорошие тройки')

    #плохие тройки:

    with open(badTripFile, 'r') as text\_file:

        for line in text\_file:

            line = line.replace('\n', '')

            splited = line.split(' ')

            triples.append((splited[1], splited[2], int(splited[3])))

    print('Считали плохие тройки')

    return triples

def generate\_image\_triples\_batch(image\_triples, batch\_size, shuffle=False):

    ''' Функция генерирует полноценную выборку с тройками из изображений

       Возвращает генератор.

   '''

    while True:

        # loop once per epoch

        if shuffle:

            indices = np.random.permutation(np.arange(len(image\_triples)))

        else:

            indices = np.arange(len(image\_triples))

        shuffled\_triples = [image\_triples[ix] for ix in indices]

        #считаем количество пакетов данных:

        num\_batches = len(shuffled\_triples) // batch\_size

        #идём циклом по пакетам данных:

        for bid in range(num\_batches):

            # loop once per batch

            images\_left, images\_right, labels = [], [], []

            #вырезаем часть данных для текущего пакета:

            batch = shuffled\_triples[bid \* batch\_size : (bid + 1) \* batch\_size]

            #формируем данные для одного пакета:

            for i in range(batch\_size):

                lhs, rhs, label = batch[i]

                images\_left.append(load\_image(lhs))

                images\_right.append(load\_image(rhs))

                labels.append(label)

            Xlhs = np.array(images\_left)

            Xrhs = np.array(images\_right)

            Y = np.array(labels)

            #print('Сформирован пакет данных #',bid)

            yield ([Xlhs, Xrhs], Y)

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

'''ПОДГОТОВКА ВЫБОРКИ И НАСТРОЙКА НЕЙРОННОЙ СЕТИ:'''

#для подсчёта времени:

start\_time = time.time()

#текущая директория:

curr\_dir = os.getcwd()

curr\_dir = curr\_dir[:curr\_dir.rfind('\\')+1] + '\\'

''' Ставим первую отклик-функцию на сохранение параметров обучения

   Файл находится в одной директории с исполянемым файлом программы!

'''

# Set up the model state, reading in prior results info if available

path\_to\_state\_file = 'states.txt'

model\_state = ModelState(path\_to\_state\_file)

callback\_list = [model\_state]

'''ЗАГРУЗКА ДАННЫХ ИЗ КОНСОЛИ:'''

imageToTrain = 1

learning\_rate = 0.01

epochs = 4

train\_size = 50000

test\_size = 10000

if len(sys.argv) > 1:

    imageToTrain = int(sys.argv[1])

    learning\_rate = float(sys.argv[2])

    epochs = int(sys.argv[3])

    train\_size = int(sys.argv[4])

    test\_size = int(sys.argv[5])

''' ПОЛУЧЕНИЕ ДАННЫХ ВЫБОРКИ:'''

IMAGE\_DIR = curr\_dir + 'datasets\\produced\_data3\\'

images = [imageToTrain] #какие картинки используются

''' 1)для обучения:'''

print('Генерация данных для обучения:')

#файлы лежат в одной директории с данной программой:

triples\_train = []

#if model\_state.state['epoch\_count'] > 0:

#    goodTripFile = 'goodHardTriples.txt'

#    badTripFile = 'badHardTriples.txt'

#    triples\_train = get\_triples\_from\_file(goodTripFile, badTripFile)

#else:

triples\_train = create\_triples(train\_size, images)

''' 2)для валидации:

       кол-во данных берётся так, чтобы:

       обучающих данных - 83%

       валидационных    - 17%

'''

print('Генерация данных для теста:')

#size = int(len(triples\_train)\*0.3/0.7)

triples\_test = create\_triples(test\_size, images)

print('Организация данных выборки проходит за:')

print("--- %s seconds ---" % (time.time() - start\_time))

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

'''ЗАГРУЗКА МОДЕЛИ НЕЙРОННОЙ СЕТИ:'''

input\_shape = (50, 50, 1)

base\_network = create\_cnn\_base\_network(input\_shape)

input\_a = Input(shape=input\_shape)

input\_b = Input(shape=input\_shape)

# because we re-use the same instance `base\_network`,

# the weights of the network

# will be shared across the two branches

processed\_a = base\_network(input\_a)

processed\_b = base\_network(input\_b)

distance = Lambda(euclidean\_distance,

                  output\_shape=eucl\_dist\_output\_shape)([processed\_a, processed\_b])

'''Инициализируем модель нейронной сети:'''

model = ''

'''Если мы только начали обучание:'''

if model\_state.state['epoch\_count'] == 0:

    model = Model([input\_a, input\_b], distance)

    opt = SGD(lr=learning\_rate, momentum=0.9)

    model.compile(loss=contrastive\_loss, optimizer=opt, metrics=[accuracy])

else:

    '''МОДЕЛЬ МЕНЯЕТСЯ ПОСЛЕ КАЖДОЙ ИТЕРАЦИИ!'''

    modelName = 'net6p\_V%d.h5' % (model\_state.state['epoch\_count'])

    print(modelName)

    model = load\_model(modelName, custom\_objects={'contrastive\_loss': contrastive\_loss,

                                                  'accuracy': accuracy})

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

'''ГЕНЕРАТОРЫ ДАННЫХ ВЫБОРКИ:'''

# Offset epoch counts if we are resuming training. If you don't do

# this, only epochs-initial\_epochs epochs will be done.

NUM\_EPOCHS = epochs

beg = model\_state.state['epoch\_count']

end = NUM\_EPOCHS + beg

BATCH\_SIZE = 128

train\_gen = generate\_image\_triples\_batch(triples\_train, BATCH\_SIZE, shuffle=True)

val\_gen = generate\_image\_triples\_batch(triples\_test, BATCH\_SIZE, shuffle=False)

#определение количества шагов на итерацию:

num\_train\_steps = len(triples\_train) // BATCH\_SIZE

num\_val\_steps = len(triples\_test) // BATCH\_SIZE

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

'''ДООБУЧЕНИЕ МОДЕЛИ:'''

start\_time = time.time()

#check: use\_multiprocessing=True

history = model.fit\_generator(train\_gen,

                              steps\_per\_epoch=num\_train\_steps,

                              callbacks=callback\_list,

                              initial\_epoch=beg,

                              epochs=end,

                              validation\_data=val\_gen,

                              validation\_steps=num\_val\_steps)

#                              max\_queue\_size=10,

#                              workers=3,

#                              use\_multiprocessing=True)

worktime = time.time() - start\_time

# compute final accuracy on training and test sets

'''Отобразить результат'''

#функция потерь:

fig, (ax1, ax2) = plt.subplots(2)

ax1.set(title = "Loss")

ax1.plot(history.history["loss"], color="r", label="train")

ax1.plot(history.history["val\_loss"], color="b", label="validation")

ax1.legend(loc="best")

#точность: на обучающей выборке и на тестовой:

ax2.set(title = "Accuracy")

ax2.plot(history.history["accuracy"], color="r", label="train")

ax2.plot(history.history["val\_accuracy"], color="b", label="validation")

ax2.legend(loc="best")

plt.tight\_layout()

#сохраним файл в директорию скриншотов:

fig.savefig(curr\_dir + 'screenWork\\net6p\_V%d\_graph.pdf' % (model\_state.state['epoch\_count']))

#plt.show()

'''Отобразить результат в консоли:'''

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

print('Дообучение на выборке:')

print('   Всего:    %d' % (len(triples\_train) + len(triples\_test)))

print('   Обучение: %d' % (len(triples\_train)))

print('   Тест:     %d' % (len(triples\_test)))

print('Проходит за:')

print("--- %s seconds ---" % (worktime))

print('\* Accuracy on training set: %0.2f%%' % (100 \* history.history["accuracy"][-1]))

print('\* Accuracy on test set:     %0.2f%%' % (100 \* history.history["val\_accuracy"][-1]))

'''Отобразить результат в файле:'''

'''\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'''

with open(curr\_dir + 'screenWork\\net6p\_V%d\_details.txt' % (model\_state.state['epoch\_count']), 'w') as f:

    print('Дообучение на выборке:', file=f)

    print('   Всего:    %d' % (len(triples\_train) + len(triples\_test)), file=f)

    print('   Обучение: %d' % (len(triples\_train)), file=f)

    print('   Тест:     %d' % (len(triples\_test)), file=f)

    print('   Время:    %s seconds' %(worktime), file=f)

    print('\nПоследние показатели обучения:', file=f)

    print('   loss:        %0.4f' % (history.history["loss"][-1]), file=f)

    print('   val\_loss:    %0.4f' % (history.history["val\_loss"][-1]), file=f)

    print('   accuracy:    %0.4f' % (history.history["accuracy"][-1]), file=f)

    print('   val\_accuracy:%0.4f' % (history.history["val\_accuracy"][-1]), file=f)

    print('\nРезультат обучения:', file=f)

    print('   Accuracy on training set: %0.2f%%' % (100 \* history.history["accuracy"][-1]), file=f)

    print('   Accuracy on test set:     %0.2f%%' % (100 \* history.history["val\_accuracy"][-1]), file=f)

model.save('net6p\_V%d.h5' % (model\_state.state['epoch\_count']), include\_optimizer=True)

#автоматическое обрезание модели:

print('Обрезание файла:')

model\_summary = str(model.to\_json())

node\_des = re.compile('sequential[\_][\d]+')

res = node\_des.search(model\_summary).group()

chopped\_model = Model(inputs=model.input[0],

                outputs=model.get\_layer(res).get\_output\_at(1))

chopped\_model.save(curr\_dir + 'choppedNets\\chopped\_net6p\_V%d.h5' % (model\_state.state['epoch\_count']))

print('Выполнено')

## compareMatching.py

Программа для сравнения точности сопоставления точек при помощи модели нейронной сети и других алгоритмов компьютерного зрения.

import cv2

import numpy as np

import random

from random import shuffle

import os

import matplotlib.pyplot as plt

import itertools

from scipy.misc import imresize

from scipy.spatial import distance

from keras.models import load\_model

from keras.models import Model # основной класс для настройки и обучения нейронной сети

from keras.layers import Input, Dense # два типа слоев нейронной сети, которые мы будем использовать

from keras.utils import np\_utils

from keras import backend as K

import pandas as pd

import scipy.spatial.distance as metric

import time

from sys import stdout

#поворот изображения:

def rotateImg(filename, fi):

    img = cv2.imread(filename)

    num\_rows, num\_cols = img.shape[:2]

    rotation\_matrix = cv2.getRotationMatrix2D((num\_cols/2, num\_rows/2), fi, 1)

    img\_rotation = cv2.warpAffine(img, rotation\_matrix, (num\_cols, num\_rows))

    #cv2.imwrite('rotImg.jpg', img\_rotation)

    return img\_rotation

#поворот координат:

def rotateXYaroundCenter(coords, fi, center):

    x, y = coords

    M = np.array(cv2.getRotationMatrix2D((center[0], center[1]), fi, 1))

    old\_coords = np.array([x, y, 1])

    new\_coords = M.dot(old\_coords)

    return new\_coords[0], new\_coords[1]

#нарисовать точки (skp и tkp - обычные списки: [[x,y], ...])

def drawKeyPoints(img, template, skp, tkp, num=-1):

    h1, w1 = img.shape[:2]

    h2, w2 = template.shape[:2]

    nWidth = w1+w2

    nHeight = max(h1, h2)

    hdif = (h1-h2)//2

    newimg = np.zeros((nHeight, nWidth, 3), np.uint8)

    newimg[hdif:hdif+h2, :w2] = template

    newimg[:h1, w2:w1+w2] = img

    maxlen = min(len(skp), len(tkp))

    if num < 0 or num > maxlen:

        num = maxlen

    #отрисовка:

    radius = 10

    for i in range(num):

        pt\_a = (int(tkp[i][0]), int(tkp[i][1]+hdif))

        pt\_b = (int(skp[i][0]+w2), int(skp[i][1]))

        newColor = np.random.randint(0,256,3)

        newColor = (int(newColor[0]), int(newColor[1]), int(newColor[2]))

        cv2.circle(newimg, pt\_a, radius, newColor)

        cv2.circle(newimg, pt\_b, radius, newColor)

        cv2.line(newimg, pt\_a, pt\_b, (newColor), 2)

    return newimg

def drawTurnedPictures(img, template):

    h1, w1 = img.shape[:2]

    h2, w2 = template.shape[:2]

    nWidth = w1+w2

    nHeight = max(h1, h2)

    hdif = (h1-h2)//2

    newimg = np.zeros((nHeight, nWidth, 3), np.uint8)

    newimg[hdif:hdif+h2, :w2] = template

    newimg[:h1, w2:w1+w2] = img

    return newimg

#показать изображение:

def show\_img(sid, img\_file, img\_title):

    plt.subplot(sid)

    plt.title(img\_title)

    plt.xticks([])

    plt.yticks([])

    plt.imshow(img\_file)

#убрать точки так, чтобы минимальное расстояние между оставшимися было dist:

#для вычисления расстояния используется евклидова метрика:

def more\_space\_for(a, dist):

    i = 0

    while i < len(a):

        j = i+1

        while j < len(a):

            if distance.euclidean(a[i], a[j]) < dist:

                a.pop(j)

            else:

                j += 1

        i += 1

    return a

'''Получить точки для анализа точности способ для сети:

       img1, img2 - две картинки для поиска точек

       для второй картинки точки получаются поворотом точек из первой

       fi - угол поворота (в градусах)

       numP - количество точек для сопоставления

'''

def getPointsFromImages(img1, img2, fi, numP, areaSize=16, min\_dist=10):

    # Initiate SIFT detector

    det = cv2.xfeatures2d.SIFT\_create()

    gray1= cv2.cvtColor(img1,cv2.COLOR\_BGR2GRAY)

    gray2= cv2.cvtColor(img2,cv2.COLOR\_BGR2GRAY)

    '''вычислим расположение точек на картинке 1 (только координаты)'''

    kp1 = det.detect(gray1,None)

    print('Первоначальное количество ключевых точек:', len(kp1))

    '''Подрезка точек: возьмём центральные:'''

    #размер области для взятия особых точек

    #возьмём только точки около центра в радиусе r = (h - areaSize)/2, где h - наименьший параметр размера изображения

    h = img1.shape[0]

    if img1.shape[0] >= img1.shape[1]:

        h = img1.shape[1]

    r = (h - areaSize)/2

    #посчитаем центр изображения:

    imgCenter = [img1.shape[1]/2, img1.shape[0]/2,]

    #берём особые точки рядом с центром из множества детектора sift:

    knew1 = [[kp1[i].pt[0], kp1[i].pt[1]] for i in range(len(kp1)) if (kp1[i].pt[0] - imgCenter[0])\*\*2 + (kp1[i].pt[1] - imgCenter[1])\*\*2 <= r\*\*2 ]

    print('Кол-во ключевых точек после краевой обрезки:', len(knew1))

    '''Подрезка точек: оставляем только те, которые находятся

      по краней мере на min\_dist расстоянии от других

   '''

    knew1 = more\_space\_for(knew1, min\_dist)

    print('Кол-во ключевых точек после удаления плотно лежащих точек:', len(knew1))

#    radius = 10

#    for p in knew1:

#        cv2.circle(img1, ((int)(p[0]),(int)(p[1])), radius, (255, 0, 0))

#

#    cv2.imshow('keypoints.jpg',img1)

#    cv2.waitKey(0)

#    cv2.destroyAllWindows()

#

    '''вычислим расположение точек на картинке 2, поворотом'''

    knew2 = knew1.copy()

    for i in range(len(knew2)):

        coords = knew2[i][0], knew2[i][1]

        n\_coords = rotateXYaroundCenter(coords, fi, imgCenter)

        knew2[i] = n\_coords

    print('Вычисление точек закончено')

    '''Вырежем изображения из картинок 1 и 2:'''

    N = areaSize//2

    mas1 = []

    mas2 = []

    #показать следующее количество точек:

    numPoints = numP if numP <= len(knew1) else len(knew1)

    for i in range(numPoints):

        #вырезаем область:

        mas1.append([gray1[(int)(knew1[i][1]-N):(int)(knew1[i][1]+N), (int)(knew1[i][0]-N):(int)(knew1[i][0]+N)], i])

        mas2.append([gray2[(int)(knew2[i][1]-N):(int)(knew2[i][1]+N), (int)(knew2[i][0]-N):(int)(knew2[i][0]+N)], i])

    #обрезаем массивы knew:

    knew1 = knew1[:numPoints]

    knew2 = knew2[:numPoints]

    print('Нарезка картинок закончена')

#    show\_img(131, mas1[9][0], "original")

#    show\_img(132, mas2[9][0], "similar")

#    plt.tight\_layout()

#    plt.show()

    '''Для чистоты проверки, перемешаем массивы:'''

    mas1 = np.random.permutation(mas1)

    mas2 = np.random.permutation(mas2)

    return knew1, knew2, mas1, mas2

def contrastive\_loss(y\_true, y\_pred):

    '''Contrastive loss from Hadsell-et-al.'06

   http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

   '''

    margin = 1

    return K.mean(y\_true \* K.square(y\_pred) +

                  (1 - y\_true) \* K.square(K.maximum(margin - y\_pred, 0)))

'''Непосредственно сопоставление точек:'''

def keypointMatching(mas1, mas2):

    '''сопоставляем точки картинок:'''

    right = 0

    wrong = 0

    skp\_final, tkp\_final = [], []

    #сделаем так, чтобы точки из массивов удалялись:

    i = 0

    count = len(mas2)

    print('Сопоставление точек:')

    while len(mas2) > 0:

        #берём дескриптор первой точки и сравниваем с дескрипторами других точек:

        all\_dists = []

        for j in range(len(mas2)):

            #получаем расстояние между дескрипторами:

            all\_dists.append(metric.euclidean(mas1[0][1], mas2[j][1]))

        #выбираем наиболее подходящее предсказание:

        #наибольшая вероятность сходства двух областей

        ind = np.argmin(all\_dists)

        #проверяем описывают ли картинки одну и ту же область:

        if mas1[0][2] == mas2[ind][2]:

            right += 1

        else:

            wrong += 1

        skp\_final.append(mas1[0][0])

        tkp\_final.append(mas2[ind][0])

        stdout.write("\rОбработана точка #%d/%d" % (i,count))

        stdout.flush()

        i += 1

        # удаляем точки из массивов skp, tkp, sd, td:

        mas1 = np.delete(mas1, 0, 0)

        mas2 = np.delete(mas2, ind, 0)

    print('\nТочность:')

    positive\_acc = (right/(right+wrong))\*100

    negative\_acc = (wrong/(right+wrong))\*100

    print('Правильные сопоставления - {0:.3f}%'.format(positive\_acc))

    print('Неправильные сопоставления - {0:.3f}%'.format(negative\_acc))

    return skp\_final, tkp\_final, positive\_acc/100

'''Сопоставление точек при помощи модели'''

def analyze\_model(knew1, knew2, img1, img2, modelName, areaSize=50):

    '''Загрузим натренированную модель:'''

    model = load\_model(modelName, custom\_objects={'contrastive\_loss': contrastive\_loss})

    gray1= cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

    gray2= cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

    '''Преобразуем точки к станлартному виду:'''

    N = areaSize//2

    mas1 = []

    mas2 = []

    for i in range(len(knew1)):

        #вырезаем область:

        mas1.append([gray1[(int)(knew1[i][1]-N):(int)(knew1[i][1]+N), (int)(knew1[i][0]-N):(int)(knew1[i][0]+N)], i])

        mas2.append([gray2[(int)(knew2[i][1]-N):(int)(knew2[i][1]+N), (int)(knew2[i][0]-N):(int)(knew2[i][0]+N)], i])

    '''Получаем дескрипторы:'''

    #для первой серии точек: mas1 (knew1)

    des1 = []

    for point in mas1:

        obj = np.array(point[0]).astype(np.float32)

        obj = obj.reshape(1, 50, 50, 1)

        obj = np.divide(obj, 255)

        des1.append(model.predict(obj)[0])

    #для второй серии точек: mas2 (knew2)

    des2 = []

    for point in mas2:

        obj = np.array(point[0]).astype(np.float32)

        obj = obj.reshape(1, 50, 50, 1)

        obj = np.divide(obj, 255)

        des2.append(model.predict(obj)[0])

    ndes1 = np.array(des1)

    ndes2 = np.array(des2)

    #Подготовим для сопоставления:

    mas1, mas2 = [], []

    for i in range(len(knew1)):

        mas1.append([knew1[i], ndes1[i], i])

        mas2.append([knew2[i], ndes2[i], i])

    '''Для чистоты проверки, перемешаем массивы:'''

    mas1 = np.random.permutation(mas1)

    mas2 = np.random.permutation(mas2)

    return keypointMatching(mas1, mas2)

'''АНАЛИЗ ТОЧНОСТИ МОДЕЛИ для алгоритмов'''

def analyze\_modelAL(detector, knew1, knew2, img1, img2, areaSize=16):

     # Initiate SIFT detector

    det = detector

    gray1= cv2.cvtColor(img1,cv2.COLOR\_BGR2GRAY)

    gray2= cv2.cvtColor(img2,cv2.COLOR\_BGR2GRAY)

    '''Преобразуем точки к станлартному виду:'''

    #массивы точек для отрисовки:

    gray1\_kp = []

    gray2\_kp = []

    for i in range(len(knew1)):

        gray1\_kp.append(cv2.KeyPoint(knew1[i][0], knew1[i][1], areaSize//2))

        gray2\_kp.append(cv2.KeyPoint(knew2[i][0], knew2[i][1], areaSize//2))

    '''вычислим дескрипторы точек на картинках '''

    desc1 = det.compute(gray1, gray1\_kp)

    desc2 = det.compute(gray2, gray2\_kp)

    #Подготовим для сопоставления:

    mas1, mas2 = [], []

    length = min(len(desc1[1]), len(desc2[1]))

    for i in range(length):

        mas1.append([knew1[i], desc1[1][i], i])

        mas2.append([knew2[i], desc2[1][i], i])

    '''Для чистоты проверки, перемешаем массивы:'''

    mas1 = np.random.permutation(mas1)

    mas2 = np.random.permutation(mas2)

    return keypointMatching(mas1, mas2)

#визуальная проверка точек:

def checkTurningPointsVisually():

    img1 = cv2.imread('a\_a1.jpg')

    img2 = rotateImg('a\_a1.jpg', 45) #поворот второй картинки

    knew1, knew2, mas1, mas2 = getPointsFromImages(img1,\

                                                   img2, \

                                                   45, \

                                                   30, \

                                                   20)

    i = 0

    j = 0

    for i in range(len(mas1)):

        for j in range(len(mas2)):

            if mas1[i][1] == mas2[j][1]:

                break

    show\_img(131, mas1[i][0], "original")

    show\_img(132, mas2[j][0], "similar")

    plt.tight\_layout()

    plt.show()

'''ОСНОВНАЯ ПРОГРАММА ПРОВЕРКИ:'''

IMAGE\_DIR = r'C:\Users\Artemij\Desktop\S\_work\evaluateBestModel\source'

def simpleMain():

    '''Инициализация картинок'''

    pictName = '1.jpg'

    picture = IMAGE\_DIR + '\\' + pictName

    angle = 20

    img1 = cv2.imread(picture)

    img2 = rotateImg(picture, angle) #поворот второй картинки

#    match3(img1, img2)

    '''Получение ключевых точек:'''

    numMatchingPoints = 400

    areaSize = 50

    min\_distance\_between\_points = 10

    knew1, knew2, mas1, mas2 = getPointsFromImages(img1,\

                                                   img2, \

                                                   angle, \

                                                   numMatchingPoints, \

                                                   areaSize, \

                                                   min\_distance\_between\_points)

    '''Отрисовать точек:'''

    pointsToDraw = 10

    imgRez = drawKeyPoints(img1, img2, knew1[:pointsToDraw], knew2[:pointsToDraw])

    cv2.imwrite('MKPtest\_{0}\_{1}\_angle\_{2}\_.jpg'.format(pictName, 'ideal', angle), imgRez)

    print('\nТестируем точность алгоритма SIFT:')

    '''проверяем точность алгоритма:'''

    skp, tkp, accuracy = [], [], 0

    det = cv2.xfeatures2d.SIFT\_create()

    skp, tkp, accuracy = analyze\_modelAL(det, knew1, knew2, img1, img2, areaSize)

    '''отрисовка точек: '''

    imgRez = drawKeyPoints(img1, img2, skp[:pointsToDraw], tkp[:pointsToDraw])

    cv2.imwrite('MKP\_{0}\_{1}\_angle\_{2}\_.jpg'.format(pictName, 'al-SIFT', angle), imgRez)

    print('\nТестируем точность алгоритма SURF:')

    '''проверяем точность алгоритма:'''

    skp, tkp, accuracy = [], [], 0

    det = cv2.xfeatures2d.SURF\_create()

    skp, tkp, accuracy = analyze\_modelAL(det, knew1, knew2, img1, img2, areaSize)

    '''отрисовка точек: '''

    imgRez = drawKeyPoints(img1, img2, skp[:pointsToDraw], tkp[:pointsToDraw])

    cv2.imwrite('MKP\_{0}\_{1}\_angle\_{2}\_.jpg'.format(pictName, 'al-SURF', angle), imgRez)

    print('\nТестируем точность модели сети:')

    '''проверяем точность модели:'''

    modelName = 'chopped\_net6p\_V12.h5'

    skp, tkp, accuracy = [], [], 0

    skp, tkp, accuracy = analyze\_model(knew1, knew2, img1, img2, modelName, areaSize)

    '''отрисовка точек: '''

    imgRez = drawKeyPoints(img1, img2, skp[:pointsToDraw], tkp[:pointsToDraw])

    cv2.imwrite('MKP\_{0}\_{1}\_angle\_{2}\_.jpg'.format(pictName, 'm-net6p', angle), imgRez)

    pass

def turnPicture():

    '''Инициализация картинок'''

    pictName = '10.jpg'

    picture = IMAGE\_DIR + '\\' + pictName

    angle = 50

    img1 = cv2.imread(picture)

    img2 = rotateImg(picture, angle) #поворот второй картинки

    img = drawTurnedPictures(img1, img2)

    cv2.imwrite('turnedPictures.jpg', img)

if \_\_name\_\_ == '\_\_main\_\_':

    simpleMain()

## hardMiningGood.py

Основной модуль для реализации поиска хороших «трудноотличимых» пар (1 вариант). Аналогично ему написан код и для майнинга плохих пар hardMiningBad.py.

from keras.models import load\_model

from keras import backend as K

import time

import os

import matplotlib.pyplot as plt

import numpy as np

import threading

from itertools import combinations

from hardMiningAux import generateAreaName, printList, reverse\_insort, writeToFile

from hardMiningAux import merging\_time

''' ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ:'''

''' Основные параметры:'''

N = 100        #кол-во точек

A = 79       #кол-во преобразований

Pgood = 10    #кол-во потоков для поиска хороших пар

lim = 25000    #сколько хор. и плох. пар мы хотим получить

pictInd = 1  #номер картинки

goodL = []   #список хороших пар типа: <val, (img1, img2, 1)>

badL = []    #список плохих пар:       <val, (img1, img2, 0)>

''' Параметры модели нейронной сети: '''

modelName = 'net6n\_V0.h5'

MODEL\_DIR = r'C:\Users\Artemij\Desktop\S\_work\NeuralNets\MyResearch\_programs'

IMAGE\_DIR = r'C:\Users\Artemij\Desktop\S\_work\NeuralNets\MyResearch\_programs\datasets\produced\_data3'

image\_cache = {}

def contrastive\_loss(y\_true, y\_pred):

    ''' Функция потерь:

       Contrastive loss from Hadsell-et-al.'06

       http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

   '''

    margin = 1

    return K.mean(y\_true \* K.square(y\_pred) +

                  (1 - y\_true) \* K.square(K.maximum(margin - y\_pred, 0)))

def load\_image(image\_name):

    ''' Возвращает изображение в формате входных данных использ. модели сети

   '''

    if not image\_cache.\_\_contains\_\_(image\_name):

        image = plt.imread(os.path.join(IMAGE\_DIR, image\_name)).astype(np.float32)

        #НОРМИРОВКА: [0:255] -> [0:1]:

        image = np.divide(image, 255)

        image\_cache[image\_name] = np.reshape(image, (1, 50, 50, 1))

    return image\_cache[image\_name]

''' ОПИСАНИЕ КЛАССОВ ПОТОКОВ'''

''' Для блокирования данных'''

threadLock = threading.Lock()

class myGoodMiningThread (threading.Thread):

    ''' Класс потока для реализации hard-mining для хороших пар:

       каждый экземпляр класса обрабатывает свою порцию данных,

       которая определяется индексом потока

   '''

    def \_\_init\_\_(self, threadID, name, pictId, \_N, \_P, modelName):

        ''' Инициализация экземпляра класса, атрибуты:

           threadID - индекс потока

           name - имя потока

           modelName - имя модели нейронной сети

               (для сравнения картинок)

       '''

        threading.Thread.\_\_init\_\_(self)

        self.threadID = threadID

        self.name = name

        self.pict = pictId

        self.model = load\_model(modelName, custom\_objects={'contrastive\_loss': contrastive\_loss})

        self.model.\_make\_predict\_function()

        ''' для хороших пар потоки разделяют работу по точкам:'''

        self.beg = (\_N//\_P)\*self.threadID

        self.end = \_N if self.threadID == \_P-1 else (\_N//\_P)\*(self.threadID + 1)

        self.gList = []

#        print("threadID  ", self.threadID)

#        print("nameThread", self.name)

#        print("pictureInd", self.pict)

#        print("model     ", self.model)

#        print("beg       ", self.beg)

#        print("end       ", self.end)

#        print("gList     ", self.gList)

    def run(self):

        ''' Основная функция работы потока

           элементы в списках находятся в ПОРЯДКЕ УБЫВАНИЯ

           1 часть: параллельная (выполняется вместе с другими потоками):

               Получение данных с помощью модели нейронной сети

           2 часть: последовательная (работает с данными, блокируя их для других потоков):

               Слияние полученных данных с данными списка GoodL

       '''

        print ("Starting " + self.name)

        for point in range(self.beg, self.end):

            for pair in combinations(range(A), 2):

                areaName0 = generateAreaName(point, self.pict, pair[0])

                areaName1 = generateAreaName(point, self.pict, pair[1])

                obj0 = load\_image(areaName0)

                obj1 = load\_image(areaName1)

                prediction = self.model.predict([obj0, obj1])

                reverse\_insort(self.gList, (prediction[0][0], (areaName0, areaName1, 1)))

        if len(self.gList) > lim:

            self.gList = self.gList[:lim]

        # Get lock to synchronize threads

        with threadLock:

            merging\_time(self.name, lim, goodL, self.gList, True)

''' ТЕСТЫ:'''

def test():

    model = load\_model(os.path.join(MODEL\_DIR, modelName), custom\_objects={'contrastive\_loss': contrastive\_loss})

    pict=1

    gList = []

    for point in range(0, 1):

            for pair in combinations(range(A), 2):

                areaName0 = generateAreaName(point, pict, pair[0])

                areaName1 = generateAreaName(point, pict, pair[1])

                obj0 = load\_image(areaName0)

                obj1 = load\_image(areaName1)

                prediction = model.predict([obj0, obj1])

                reverse\_insort(gList, (prediction, (areaName0, areaName1, 1)))

    if len(gList) > lim:

        gList = gList[:lim]

    printList(gList)

def goodBruteForceTest(filename, varLimit):

    ''' Тест проверяет, можно ли вставить в полученный список еще элементы

       Элементы берутся перебором областей рассматриваемых изображений

       (тест написан для проверки при небольшом кол-ве точек (2))

   '''

    model = load\_model(os.path.join(MODEL\_DIR, modelName), custom\_objects={'contrastive\_loss': contrastive\_loss})

    pict=1

    bestList = []

    file = open(filename, 'r')

    for line in file:

        line = line.replace('\n', '')

        splited = line.split(' ')

        bestList.append((float(splited[0]), (splited[1], splited[2], int(splited[3]))))

    file.close()

    print('Лучший список:')

    printList(bestList)

    gList = []

    i\_lim = 0

    for point in range(0, 2):

        for pair in combinations(range(A), 2):

            areaName0 = generateAreaName(point, pict, pair[0])

            areaName1 = generateAreaName(point, pict, pair[1])

            obj0 = load\_image(areaName0)

            obj1 = load\_image(areaName1)

            prediction = model.predict([obj0, obj1])[0][0]

            if bestList[-1][0] > prediction and \

               (prediction, (areaName0, areaName1, 1)) in bestList:

                print('Что-то не так: в список попал непонятный элемент')

                print((prediction, (areaName0, areaName1, 1)))

            elif bestList[-1][0] <= prediction and \

                 (prediction, (areaName0, areaName1, 1)) not in bestList:

                print('Что-то не так: в список не попал хороший элемент')

                print((prediction, (areaName0, areaName1, 1)))

            i\_lim += 1

            if i\_lim > varLimit:

                break

        if i\_lim > varLimit:

            break

    if len(gList) > lim:

        gList = gList[:lim]

    printList(gList)

'''ОСНОВНАЯ ФУНКЦИЯ:'''

if \_\_name\_\_ == '\_\_main\_\_':

    '''Поиск хороших пар:'''

    threads = []

    for i in range(Pgood):

        threads.append(myGoodMiningThread(i, \

                                          'Thread-%d' % (i), \

                                          pictInd, \

                                          N, \

                                          Pgood, \

                                          os.path.join(MODEL\_DIR, modelName)))

    print ("%s: %s" % ('Starting threads', time.ctime(time.time())))

    for thread in threads:

        thread.start()

    # Wait for all threads to complete

    for t in threads:

       t.join()

    print ("Exiting Main Thread")

    path = r'C:\Users\Artemij\Desktop\S\_work\NeuralNets\MyResearch\_programs'

    source\_filename = 'goodHardTriples.txt'

    writeToFile(path, source\_filename, goodL)

#    printList(goodL)