РЕФЕРАТ

Магистерская диссертация содержит 76 страниц, 32 рисунка, 8 таблиц, 14 использованных источников.

МАШИННОЕ ОБУЧЕНИЕ, ЗАДАЧА РЕГРЕССИИ, АНАЛИЗ ДАННЫХ, ГЛУБОКОЕ ОБУЧЕНИЕ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, АНАЛИЗ ИЗОБРАЖЕНИЙ, СОВМЕЩЕНИЕ ИЗОБРАЖЕНИЙ, МИКРОСКОПИЯ.

Магистерская диссертация посвящена методике высокоточного совмещения отдельных фрагментов панорамных мозаичных изображений в единое изображение с использованием искусственных нейронных сетей. Подобные изображения получаются в результате сканирующей съёмки методами электроннолучевой микроскопии относительно крупных исследуемых образцов. Методика предполагает запрет на внесение корректировок в исходные кадры, несмотря на возможное наличие некоторых шумов и искажений в исходных данных.

Реализована модель сверточной нейронной сети. Обучение и тестирование модели, сбор и генерация данных реализованы в виде отдельной программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

ОСНОВНАЯ ЧАСТЬ 7

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 8

1.1 Применение машинного обучения 8

1.1.1 Постановка задачи регрессии 8

1.1.2 Решение задачи регрессии 8

1.2 Нейронные сети 11

1.2.1 Простейшая нейронная сеть 11

1.2.2 Глубокие нейронные сети 12

1.3 Применение нейронных сетей для анализа изображений 16

1.3.1 Сверточные нейронные сети 17

2 ПРАКТИЧЕСКАЯ ЧАСТЬ 20

2.1 Описание задачи 20

2.2 Постановка задачи 21

2.3 Описание данных 24

2.3.1 Особенности данных 26

2.4 Методы решения 30

2.4.1 Известные методы совмещения 30

2.4.2 Предлагаемый метод 37

2.4.3 Сбор данных 38

2.5 Нейросетевой подход 46

2.5.1 Эволюция модели 47

2.5.2 Итоговая модель нейронной сети 50

2.5.3 Примеры работы нейронной сети 52

2.6 Разработка программного обеспечения 59

ЗАКЛЮЧЕНИЕ 61

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 62

ПРИЛОЖЕНИЯ..... 64

ВВЕДЕНИЕ

Современные методы электронной микроскопии позволяют получать изображения исследуемых объектов с разрешением в несколько нанометров на пиксель [1]. Как правило такие изображения имеют растровый формат в градациях серого. Размер одного получаемого на современном оборудовании кадра обычно не превышает 10000×10000 пикселей. Таким образом, если необходимо получить изображение какого-либо относительно крупного образца с достаточно высоким разрешением, то приходится применять последовательную покадровую съёмку. В результате чего получается упорядоченный комплект отдельных кадров, который является мозаичным изображением исследуемого образца.

Как правило, современное оборудование позволяет производить покадровую съёмку относительно больших образцов в автоматическом режиме. Размер одного кадра, шаг одного кадра, обеспечивающий требуемое взаимное перекрытие, устанавливается оператором в начале процесса съёмки. После этого съёмка производится автоматически с достаточно высокой точностью. Однако, по различным физическим и технологическим причинам, получаемые кадры могут иметь как малые локальные искажения и шумы, так и значительные дефекты, влияющие на большую часть кадра. Кроме того, некоторое оборудование для сканирующей электронной микроскопии в принципе не может обеспечить требуемую точность разбиения на кадры с установленным смещением.

Вследствие таких обстоятельств возникает потребность в специальных методиках и инструментах для высокоточного совмещения полученных кадров в единое ровное изображение. Необходимо с одной стороны скорректировать взаимное расположение кадров таким образом, чтобы максимальным образом компенсировать имеющиеся дефекты и искажения в областях их стыковки, а с другой стороны полностью сохранить исходное изображение, полученное в результате электронного сканирования.

Так как описанная задача имеет чрезвычайно высокую актуальность в области производства и исследования микроэлектронных компонентов [2, 3], то именное такие изображения являлись модельными в ходе данного исследования.

ОСНОВНАЯ ЧАСТЬ

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Применение машинного обучения

В данном разделе будет приведена теоретическая информация о методах машинного обучения, применённых в диссертации.

1.1.1 Постановка задачи регрессии

Определение 1.1. [4]. Обучающая выборка — выборка, по которой производится настройка (оптимизация параметров) модели зависимости.

Пусть задана обучающая выборка $S = ((\overline{x}_1, y_1), ..., (\overline{x}_l, y_l))$, где $\overline{x}_i \in Ў^n, y_i \in Ў$ при $i = 1, K, l$. Тогда задача простой линейной регрессии формулируется следующим образом.

Задача 1.1. [5]. Найти такую линейную функцию

$$f(\overline{x}) = (\overline{w} \cdot \overline{x}) + b, \tag{1.1}$$

которая наилучшим образом интерполирует элементы выборки *S*.

1.1.2 Решение задачи регрессии

Задача 1.1, поставленная в подразделе 1.1.1 уже была решена Гауссом и Лежандром в XVIII веке с помощью минимизации суммы квадратов разностей значений функции $f(\overline{x}_i)$ и точек y_i.

Обозначим произвольный вектор $\overline{x}$ через матрицу (вектор-столбец) размера $(n \times 1)$

$$\overline{x} = \begin{pmatrix} x_1 \\ x_2 \\ M \\ x_n \end{pmatrix}.$$

Этот же транспонированный вектор $\bar{x}^T$ $(1\times n)$ может быть записан в виде строки $\bar{x}^T = (x_1,...,x_n)$.

Согласно методу наименьших квадратов необходимо минимизировать квадратичную функцию потерь, которая имеет вид

$$L(\bar{w},b) = \sum_{i=1}^{l}(y_i - f(\bar{x}_i))^2. \quad (1.2)$$

Подставив (1.1) в (1.2)

$$L(\bar{w},b) = \sum_{i=1}^{l}(y_i - (\bar{w}\cdot\bar{x}_i) - b)^2. \quad (1.3)$$

Через $\tilde{w}$ обозначается расширенный вектор-столбец весовых коэффициентов и свободного члена

$$\tilde{w} = \begin{pmatrix} w_1 \\ w_2 \\ \text{M} \\ w_n \\ b \end{pmatrix}.$$

А также расширенный вектор-столбец переменных $\tilde{x}$:

$$\tilde{x} = \begin{pmatrix} x_1 \\ x_2 \\ \text{M} \\ x_n \\ 1 \end{pmatrix}.$$

Тогда функция регрессии (1.1) принимает вид:

$$f(\tilde{x}) = (\tilde{w}\cdot\tilde{x}) \quad (1.4)$$

Рассмотрим матрицу размера $(l\times(n+1))$, строками которой являются расширенные векторы-строки переменных $\tilde{x}^T = (\bar{x}^T,1)$.

$$\hat{X} = \begin{pmatrix} \bar{x}_1^T \\ \bar{x}_2^T \\ \text{M} \\ \bar{x}_l^T \end{pmatrix} = \begin{pmatrix} x_{11},\ldots,x_{1n},1 \\ x_{21},\ldots,x_{2n},1 \\ \text{M} \\ x_{l1},\ldots,x_{ln},1 \end{pmatrix}.$$

Также вводится вектор-столбец значений интерполяции:

$$\bar{y} = \begin{pmatrix} y_1 \\ y_2 \\ \text{M} \\ y_l \end{pmatrix}.$$

Определение 1.2. Остатками называются разности вида $|f(\bar{x}_i) - y_i|$. Вектор столбец остатков имеет вид $\bar{y} - (\hat{X}\hat{w})$.

Тогда (1.3) можно записать как квадрат нормы вектора-столбца остатков:

$$L(w) = \left\| \hat{X}\hat{w} - \bar{y} \right\|^2 = (\bar{y} - \hat{X}\hat{w})^T (\bar{y} - \hat{X}\hat{w}), \tag{1.5}$$

где норма произвольного вектора x длиной n есть $\|x\| = \sum_{i=1}^{n} |x_i|$.

Задача регрессии может быть записана в виде задачи минимизации квадрата нормы вектора остатков:

$$L(w) = \left\| \hat{X}\hat{w} - \bar{y} \right\|^2 \to \min. \tag{1.6}$$

Для поиска минимума (1.6) частные производные по переменным $w_1,\ldots,w_n,b$ приравниваются к нулю. Образуется система из $n+1$ уравнений

$$\frac{\partial L(\hat{w})}{\partial \hat{w}} = -2\hat{X}^T\bar{y} + 2\hat{X}^T\hat{X}\hat{w} = \bar{0}.$$

Система приводится к виду

$$\hat{X}^T\hat{X}\hat{w} = \hat{X}^T\bar{y}.$$

Тогда решением задачи 1.1. является следующее уравнение

$$\hat{w} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\bar{y}. \tag{1.7}$$

1.2 Нейронные сети

За последние годы, ввиду роста вычислительных мощностей, стало активно развиваться применение нейросетевого подхода к решению нетривиальных задач: обработка текста, изображений, аудио- и видеофайлов. Порой нейронные сети оказываются «умнее» людей, например, обыгрывая их в шахматы, или создавая музыку. Поэтому в данной работе было решено исследовать возможность применения нейронных сетей к решению задачи совмещения изображений.

В этом разделе речь пойдет о нейронных сетях. Будут рассмотрены аспекты, которые применялись в данной диссертации.

1.2.1 Простейшая нейронная сеть

В 1958 году Фрэнк Розенблатт представил модель линейного перцептрона [13]. Внутри этой модели находится математический аппарат, который был описан в подразделах 1.1.1 и 1.1.2. На основе этой абстракции и построены современные нейронные сети.

Перцептрон представлен на рис. 1.1. Он состоит из входных сигналов (входной слой - синие кружки), весовых коэффициентов (зеленые прямоугольнки), сумматора входных сигналов (оранжевый кружок), функции активации (зеленый кружок), и выходного слоя (Y).

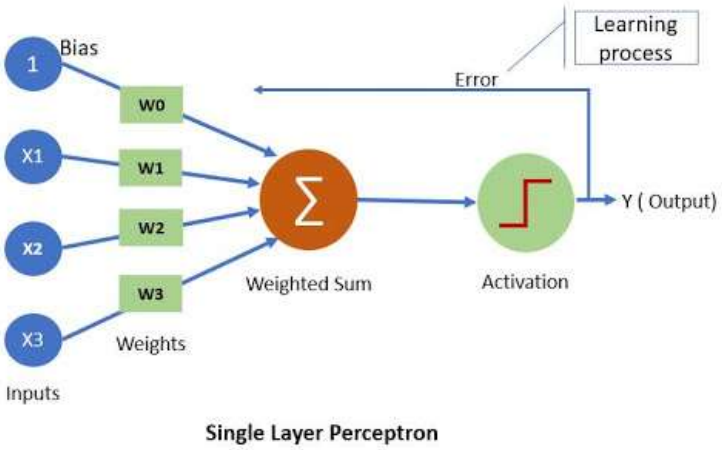


Рис. 1.1 Схема перцептрона Розенблатта

1.2.2 Глубокие нейронные сети

Можно видеть, что входным слоем является расширенный вектор-столбец переменных x (из раздела 1.1.2), который умножается на расширенный вектор-столбец весовых коэффициентов и свободного члена w. Далее идет суммирование результата произведения. После этого сумма проходит через функцию активации, из чего получается предсказываемое значение y.

Описанный абзацем выше процесс называется прямым распространением ошибки. Далее происходит корректировка весов методом градиентного спуска, что в свою очередь носит название обратное распространение ошибки. Этот процесс повторяется до тех пор, пока веса не скорректируются таким образом, что сеть начнет предсказывать результат с желаемой точностью.

Перед обучением сети задают количество эпох (сколько раз сеть совершит прямое и обратное распространение ошибки) и коэффициент обучение η, который отвечает за скорость обучения модели. На каждом эпохе рассчитываются значения, на которые необходимо скорректировать текущие веса, по следующей формуле:

$$\Delta w = -\eta \frac{\partial L(w)}{\partial w}. \tag{1.8}$$

Таким образом значение весов на эпохе t можно записать следующим образом: $w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)}$. Значения w_{ij} на первой эпохе формируются случайным образом.

Перцептрон также называют искусственным нейроном. Из них образуются нейронные сети.

Когда решалась задача линейной регрессии, функции активации не требовалась. В реальности же перцептроны не могут быть линейным, потому что сеть не сможет нормально обучиться. Поэтому, когда в сетях фигурируют скрытие слои, то на их выход накладывают функцию активации.

Функций активации достаточно большое количество. Самые популярные из них приведены на рис. 1.2.

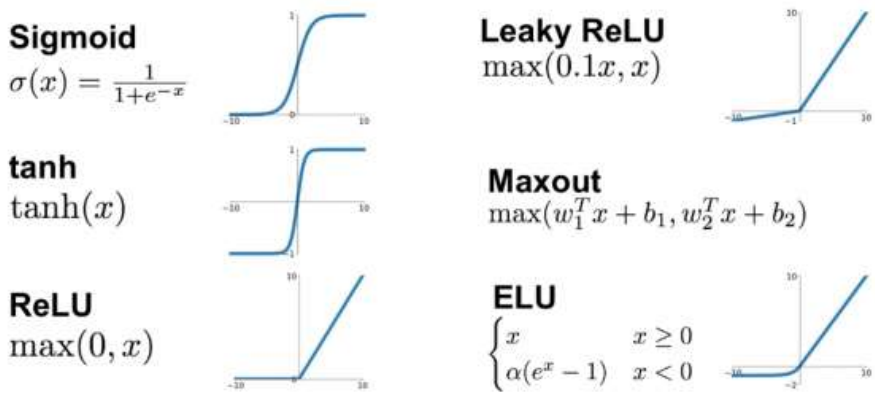


Рис. 1.2 Наиболее распространенные функции активации [14]

Чаще всего используют сигмоид, но в задачах классификации. В данной диссертации решается задача регрессии, поэтому применяемые в работе функции активации – ReLU и Leaky ReLu.

Таким образом можно резюмировать, что нейронная сеть состоит из:

- искусственных нейронов (перцептронов Розенблатта);
- слоёв, которые образуются из нейронов;
- функций активаций, которые применяются к выходам слоёв или нейронов.

В свою очередь слои в нейронных сетях бывают трех видов:

- входной;
- скрытый;
- выходной.

На рис. 1.3. представлена нейронная сеть с одним скрытым слоем. Синими кружками является входной слой, зелеными – скрытый слой, а выходной слой состоит из одного нейрона – красный кружок.

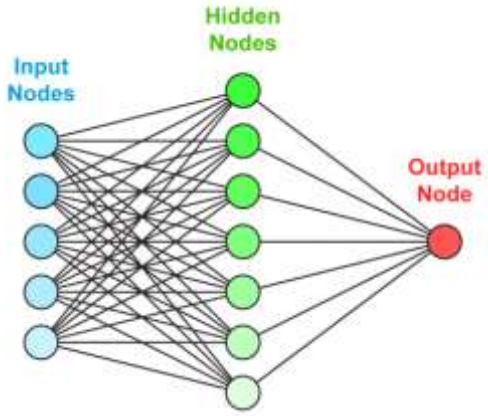


Рис. 1.3 Нейронная сеть со скрытым слоем [14]

Когда у нейронной сети есть один и более скрытых слоев, то ее называют глубокой. Когда слоев больше одного, то каждый последующий слой будет считать входными сигналами выход с предыдущего скрытого слоя.

Глубокие сети прямого распространения, которые называют также нейронными сетями прямого распространения, или многослойными перцептронами (МСП), – самые типичные примеры моделей глубокого обучения [6]. «Глубина» сети измеряется количеством скрытых слоев.

Определение 1.3. «Прямое распространение» означает, что распространение информации начинается с x, проходит через промежуточные вычисления, необходимые для определения $f(x)$, и заканчивается выходом y.

Такие сети пытаются более абстрактно понимать входные данные. Они ищут зависимости, извлекают и преобразовывают признаки из данных. Уже

известное большое количество популярных глубоких нейронных сетей, они все решают разные задачи. Иногда удается комбинировать или модифицировать архитектуры, собирать ту сеть, которая подойдет под конкретную задачу.

Разновидности популярных глубоких нейронных сетей приведены на рис. 1.4. За основу в данной работе была взята архитектура DCN.

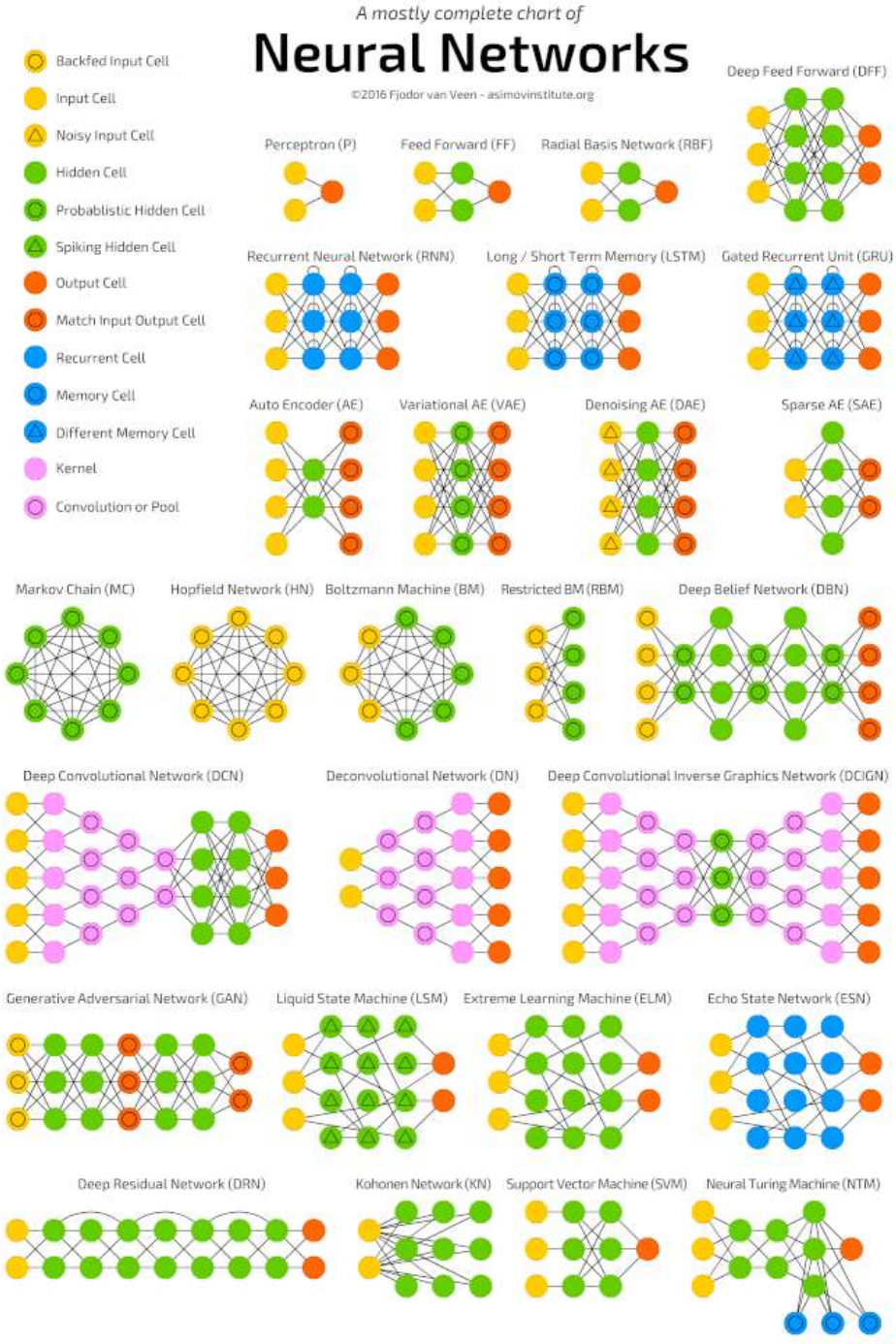


Рис. 1.4 Наиболее распространненые архитектуры нейронных сетей [14]

1.3 Применение нейронных сетей для анализа изображений

Как было описано выше – нейронные сети в последние годы хорошо себя зарекомендовали в работе с изображениями. Как раз операции с изображениями являются целевой задачей данной работы. Графическое представление информации - один из самых распространенных ее видов, поэтому на текущий момент существует огромное количество работ, посвященных работе с изображениями. А также сформулировано множество задач, для которых изображения являются входными данными:

- детектирование, локализация объектов;
- генерация изображений;
- слияние стилей изображений;
- классификация изображений по различным признакам;
- и множество других.

Изображения – удобный формат для обработки нейронными сетями. Изображение можно представить в виде тензора с размерностями $m \times n \times 1$ (изображения в градациях серого, именно такие являются модельными в практической части), $m \times n \times 3$ (RGB изображения), $m \times n \times 4$ - (RGB изображения с каналом прозрачности).

Элементом таких тензоров является значение интенсивности пикселя, которое принимает все целые значения в диапазоне от 0 до 255. 0 – минимальная интенсивность, черный цвет, 255 – максимальная интенсивность, белый цвет.

В случае изображения в градациях серого (монохромные изображения), можно сказать, что это матрица размера $m \times n$, элементами которой являются значение интенсивности пикселей.

RGB изображение можно разложить в три монохромных. Размерность 3 образуется из красного (R – red), зеленого (G – green) и синего (B – blue)

каналов. Каждый по отдельности есть монохромное изображение. Но вместе они образуют цветное изображение.

Экраны современных устройств устроены таким образом, что минимальной «ячейкой» экрана является физический пиксель. В свою очередь он состоит из трех, грубо говоря, лампочек – красного, зеленого и синего цветов. Регулируя их интенсивность, появляется возможность получить $256 \times 256 \times 256 = 16,7$ миллионов цветов. Декомпозицию фрагмента цветного изображения можно наблюдать на рис. 1.5.

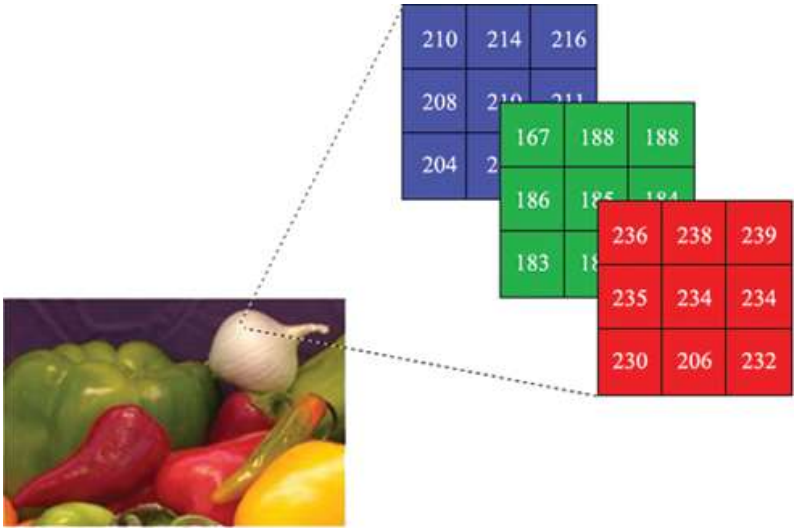


Рис. 1.5 Декомпозиция по каналам фрагмента цветного RGB-изображения

Перечисленные выше факты дают широкие возможности для обработки изображений нейронными сетями. Чаще всего при работе с графическими данными используются сверточные нейронные сети.

1.3.1 Сверточные нейронные сети

Рассматриваемые ранее модели нейронных сетей имели структуры с полносвязными слоями, то есть вектор-столбец входных сигналов умножался на вектор-столбец весовых коэффициентов, а после результат обрабатывался функцией активации. Эта процедура повторялась от слоя к слою.

Однако, изображения имеют несколько иную структуру, в них присутствуют очертания, текстуры, фигуры, контуры. Поэтому необходима модель, способная принимать во внимание вышеперечисленные факторы.

Сверточная нейронная сеть основана на математической операции свертки.

Двумерная свертка (2D Convolution) начинается с задания матрицы весов (аналог весовых коэффициентов, которые фигурировали в предыдущих разделах). Такая матрица называется ядром свёртки.

Ядро итеративно проходится по двумерному изображению, поэлементно выполняет операцию умножения с той частью входных данных, над которой оно находится. Затем происходит суммирование полученных значений в один выходной пиксель. Такой подход позволяет локально выделить признаки на изображении вне зависимости от их расположения. Процедура проиллюстрирована на рис. 1.6.

Ядро производит вышеописанную процедуру над всеми частями входных данных, над которыми оно появлялось в процессе прохождения над всем изображением. На выходе образуется двумерная матрица признаков.

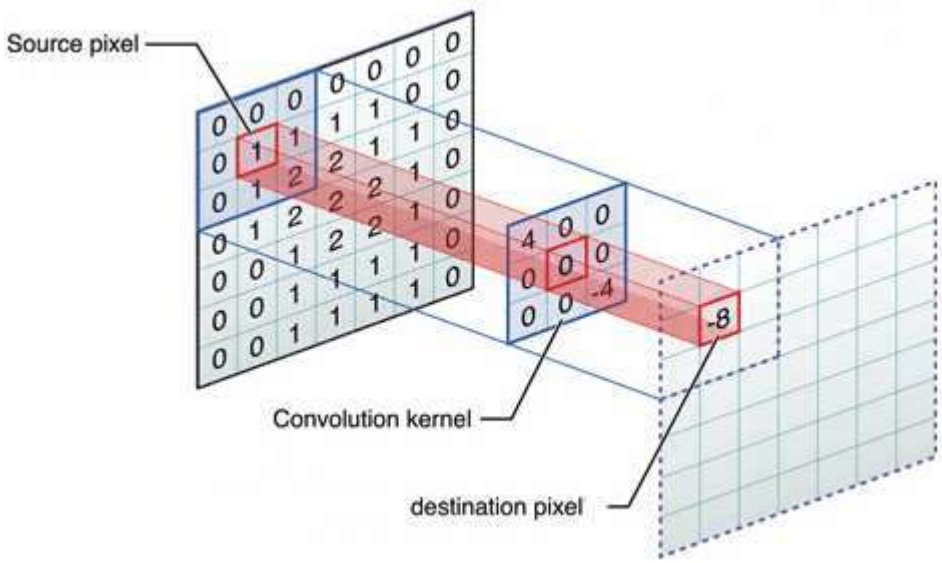


Рис. 1.6 Операция двумерной свертки

Каждым ядром формируются карта признаков, которая представляет собой выход сверточного слоя, а также являются входным слоем для последующего.

Следующим шагом является применение функции активации к карте признаков. Чаще всего в моделях сверточных сетей используют ReLU.

Далее идет слой субдискретизации (слой пулинга), на котором происходит нелинейное уплотнение карты признаков. Группа пикселей чаще всего размера 2×2 нелинейным преобразованием уплотняется до одного пикселя. Наиболее популярная функция – функция максимума, при котором преобразование проходит по всем непересекающимся прямоугольникам или квадратам из карты признаков и ужимает их до одного пикселя, который имел максимальное значение. Наблюдать данную процедуру можно на рис. 1.7.

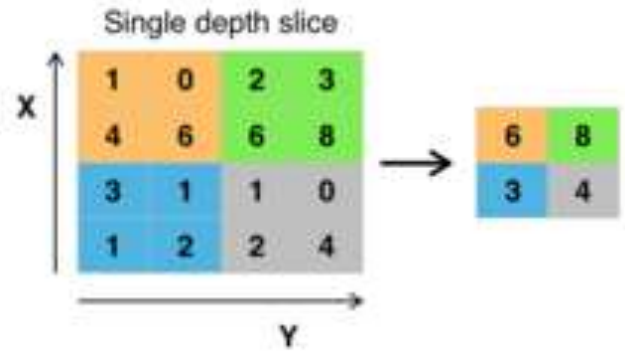


Рис. 1.7 Результат работы слоя субдискретизации с функцией максимума 2×2

Данная процедура позволяет уменьшить пространственный объем изображения. В её основе лежит идея, которая утверждает, что если на предыдущей операции свертки уже были найдены некоторые признаки, то для дальнейшей обработки столь подробные изображения уже не необходимы и производится уплотнение. А также фильтрация лишних деталей помогает сети не переобучаться.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Практическим результатом работы является обученная модель сверточной нейронной сети, способная предсказать необходимые значения сдвига по осям X и Y одного изображения относительно другого для точного совмещения. Также было спроектировано прикладное программное обеспечение, позволяющее проектировать, обучать и тестировать модели нейронных сетей.

Отдельно были реализованы программные компоненты решающие задачи сбора и аугментации тренировочного набора данных.

2.1 Описание задачи

В некоторых прикладных задач требуется собрать цельный панорамный снимок на основе материала, отснятого кусками.

Например, при съемке микроскопом или при аэросъемке. Такая задача называется задачей сшивки.

Рассмотрим задачу при следующих ограничениях:

- высокие требования к точности совмещения изображений: максимальное отклонение по одной из осей сдвига 1-2 пикселя;
- кадры изображения имеют одинаковый масштаб и являются прямоугольными, каждая область перекрытия также является прямоугольной;
- кадры получены одной камерой, но в разные моменты времени и/или при различных параметрах съёмки;
- допускается наличие шумов, искажения и дефектов, обусловленных различными факторами на этапах съёмки;
- входные данные имеют широкое разнообразие по характеристикам;
- нельзя искажать исходные изображения.

Задача заключается в том, что необходимо как можно точнее сопоставить два смежных черно-белых цифровых изображения, у которых есть общая область перекрытия, т.е. разница интенсивности каждой пары сопоставленных пикселов была как можно меньше.

2.2 Постановка задачи

Пусть матрица $IM = \begin{pmatrix} im_{11} & \text{K} & im_{1n} \\ \text{M} & \text{O} & \text{M} \\ im_{m1} & \text{L} & im_{mn} \end{pmatrix}$ - массив исходных изображений, полученных с микроскопа, где im_{st} экземпляр одного исходного кадра.

Кадр im_{st} представляет собой изображение в градациях серого размерами $l \times k$, где k – ширина изображения, а l – высота. im_{st} также можно представить в виде матрицы $im = \begin{pmatrix} p_{11} & \text{K} & p_{1k} \\ \text{M} & \text{O} & \text{M} \\ p_{l1} & \text{L} & p_{lk} \end{pmatrix}$, где $p = 0,1,...,255$ - интенсивность пиксела изображения.

Между двумя соседними кадрами из IM гарантированно существует область перекрытия S. Для двух соседних горизонтальных кадров ее размер равен $l \times s$, для двух вертикальных $s \times k$.

Значение s задается еще на стадии съёмки панорамы. То есть при съемке один кадр накладывается на другой на s пикселов. Но из-за особенностей съемки идеального результата достичь практически невозможно, появляются искажения, сдвиги, шумы, что приводит к неточностям.

Введем метрику схожести двух изображений в градациях серого.

Определение 2.1 Рассмотрим два изображения $L = \begin{pmatrix} l_{11} & \text{K} & l_{1n} \\ \text{M} & \text{O} & \text{M} \\ l_{m1} & \text{L} & l_{mn} \end{pmatrix}$ и $R = \begin{pmatrix} r_{11} & \text{K} & r_{1n} \\ \text{M} & \text{O} & \text{M} \\ r_{m1} & \text{L} & r_{mn} \end{pmatrix}$ с одинаковой размерностью $m \times n$. Метрикой схожести двух изображений будем называть следующую функцию:

$$D = \frac{1}{nm} \sum_{i=0}^{n} \sum_{j=0}^{m} (l_{ij} - r_{ij})^2 \tag{2.1}$$

При идеальном совпадении изображений $D = 0$. То есть интенсивности каждого пикселя из изображения L в точности совпадают с интенсивностями каждого пикселя из изображения R на той же самой позиции. Визуализация данной метрики представлена на рис. 2.10 в разделе 2.4.

Рассмотрим два соседних кадра im_{st} и $im_{(s+1)t}$. Для удобства обозначим их L и R соответственно. Выделим на них области, которые должны перекрыться при наложении кадров. Область перекрытия будет иметь размеры $m \times s$. Её также можно представить в виде матрицы $Z = \begin{pmatrix} z_{11} & \text{K} & z_{1s} \\ \text{M} & \text{O} & \text{M} \\ z_{m1} & \text{L} & z_{ms} \end{pmatrix}$.

При наложении у левого кадра будет перекрыт правый край на s пикселов. А у правого левый край соответственно. Выделим из левого и правого кадров подматрицы, которые накладываются друг на друга.

Для левого изображения $L[k:(k-s),l]$, для правого изображения $R[0:s,l]$. Визуально можно наблюдать выделение подматриц на рис. 2.1.

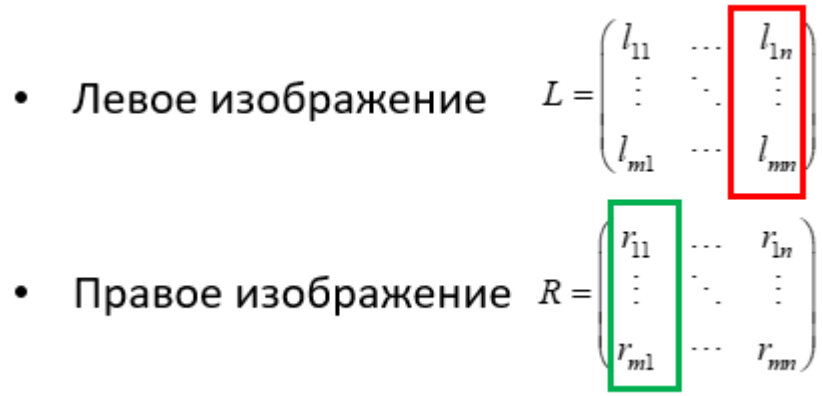


Рис. 2.1 Выделение подматриц из левого и правого изображений

Таким образом образовались две матрицы, назовем их $LZ = \begin{pmatrix} lz_{11} & \text{K} & lz_{1s} \\ \text{M} & \text{O} & \text{M} \\ lz_{l1} & \text{L} & lz_{ls} \end{pmatrix}$ и $RZ = \begin{pmatrix} rz_{11} & \text{K} & rz_{1s} \\ \text{M} & \text{O} & \text{M} \\ rz_{l1} & \text{L} & rz_{ls} \end{pmatrix}$ соответственно. Размерность у этих матриц будет $s \times l$.

Так как матрицы одинаковой размерности, для них можно рассчитать метрику схожести двух изображений. На практике эта метрика вряд ли будет равна нулю. Из-за проблем, которые возникают в процессе съемки, появляются различные дефекты, которые будут описаны далее в диссертации. Поэтому, чтобы наилучшим образом сопоставить два изображения, необходимо двигать одно относительно другого в поиске минимальной метрики схожести.

Для этого «фиксируется» левое изображение, а правое сдвигается на $\Delta i, \Delta j \in$ ў .

Таким образом формулируется задача:

Задача 2.1 Необходимо найти такие $\Delta i, \Delta j$, при которых значение метрики схожести двух изображений *D* будет минимальным:

$$D = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (lz_{ij} - rz_{i+\Delta i, j+\Delta j})^2 \to \min \qquad (2.2)$$

Для совмещения изображений с горизонтальным перекрытием (верхний и нижний кадры) производятся все те же самые процедуры, меняются лишь размерности матриц на противоположные.

2.3 Описание данных

В данной диссертации за основу были взяты исходные данные с электронного микроскопа. Но для решаемой задачи существуют и другие подходящие наборы данных, например – аэросъемка.

Данные представляют собой изображения в градациях серого размером $m \times n$ (в основном 2000×2000, но бывают исключения) пикселей со взаимной областью перекрытия в $s = 150$ пикселей. Совмещая исходные изображения, получается панорама исследуемого объекта. Абстрактный пример приведен на рис. 2.2.

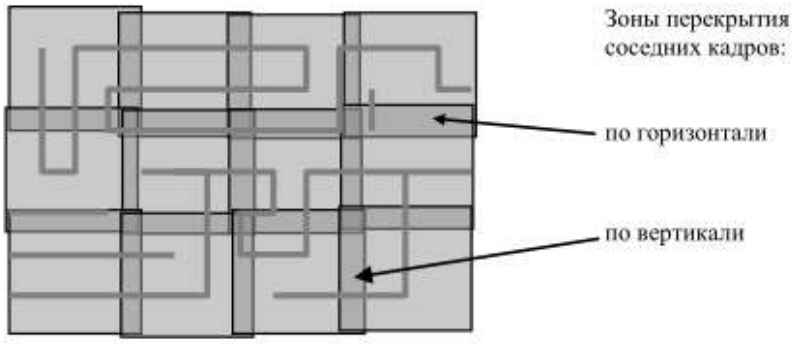


Рис. 2.2 Пример совмещения исходных изображений в итоговую панормау

Если же переходить к более реальным примерам, то как выглядят два отдельно взятых кадра из общей панорамы можно видеть на рис. 2.3. Левый – изображение под номером 1. Правый изображение под номером 2.

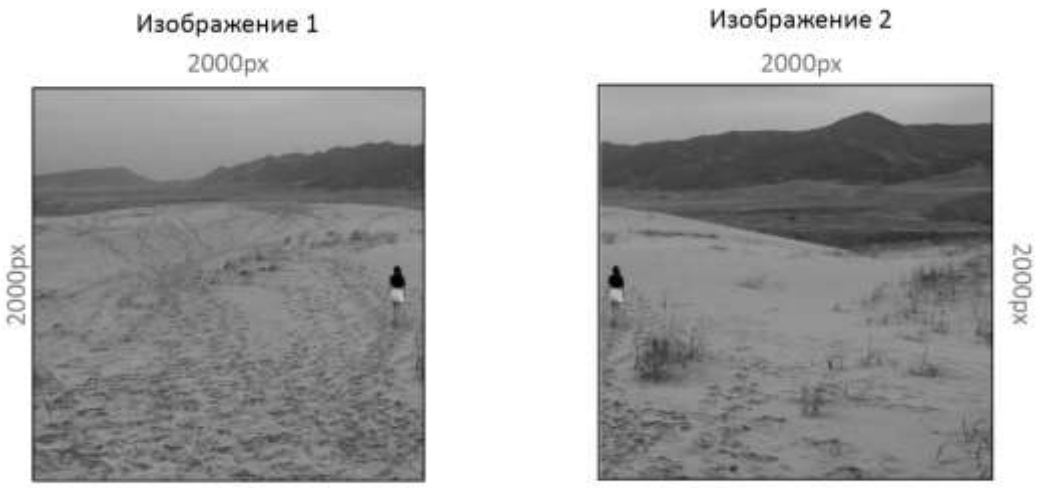


Рис. 2.3 Пример двух смежных изображений

Как можно видеть – и на левом, и на правом кадрах присутствует один и тот же объект, таким образом можно наблюдать область перекрытия.

В примере, приведенном выше, для человека очевидно, как совместить два этих изображения. В действительности на практике немного иная ситуация: исходные изображения зачастую имеют регулярную структуру, как можно наблюдать на рис. 2.4.

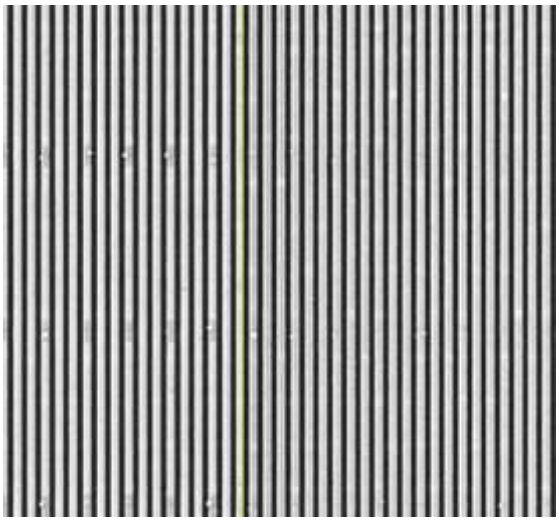


Рис. 2.4 Пример области перекрытия кадров с регулярными структурами. Жёлтая линия обозначает границу раздела двух соседних кадров

2.3.1 Особенности данных

В процессе съёмки поверхности некоторого исследуемого объекта методом сканирующий электронной микроскопии формируются растровые изображения в градациях серого. На контрастность, разрешающую способность, зашумленность, наличие сбоев, наличие астигматизма и другие параметры оказывают прямое или косвенное влияние как физические характеристики компонентов электронного микроскопа, так и, в некоторой степени, параметры окружающей среды.

Физические характеристики оборудования обеспечивают воспроизведение результатов проводимых экспериментов в установленном диапазоне параметров. При этом, естественным образом возникает погрешность в получаемых данных. Как правило, такая погрешность тем выше, чем ближе устанавливаемые параметры проведения эксперимента находятся к пределу физических характеристик используемого оборудования. Иными словами, если съёмка исследуемого образца производится с параметрами, близкими к предельным характеристикам используемого микроскопа, то тем выше будет зашумленность и/или локальные отличия при повторной съёмке одной и той же области исследуемого образца при одних и тех же параметрах съёмки (рис. 2.5).

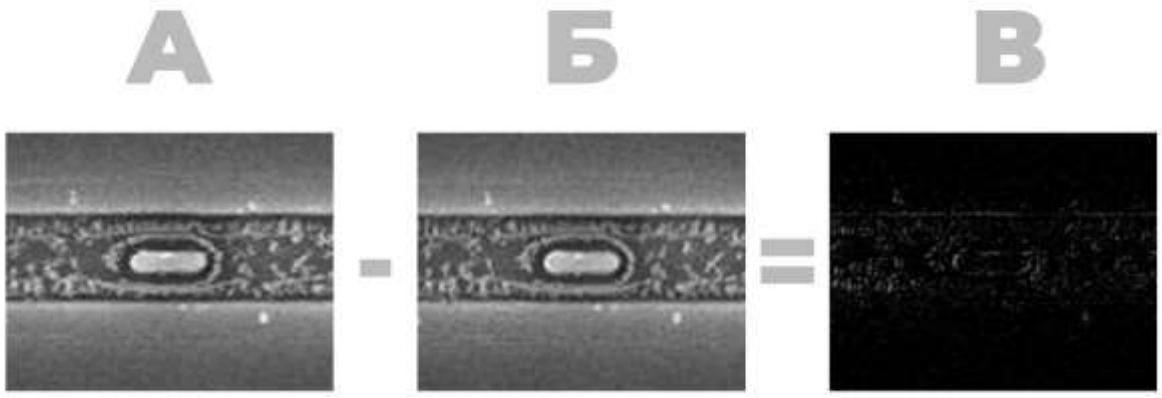


Рис. 2.5 Демонстрация локальных различий при повторной съёмке одной и той же области исследуемого образца: А - первый вариант съёмки, Б -

второй вариант съёмки, В - карта отличий кадров А и Б, полученная методом попиксельной разницы

Некоторые режимы сканирования сами по себе предполагают наличие значительного мелкозернистого шума (рис. 2.6).

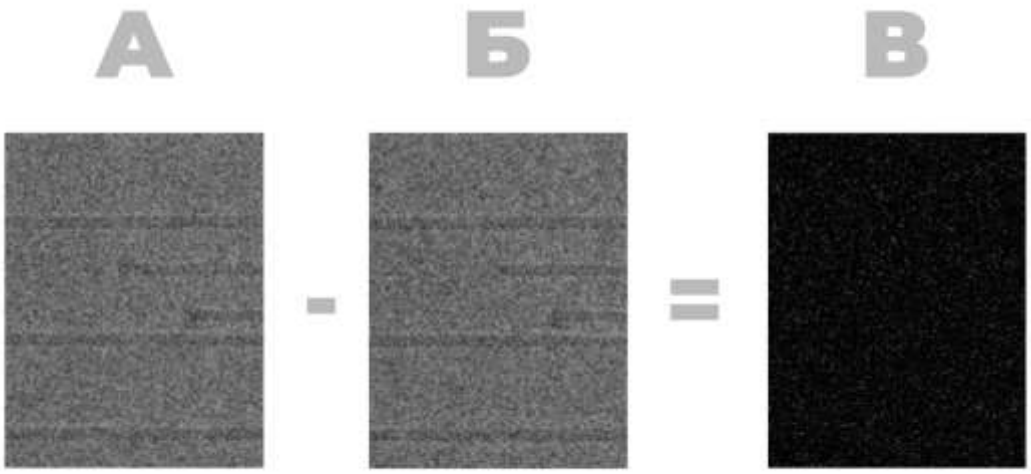


Рис. 2.6 Демонстрация влияния мелкозернистого шума при повторной съёмке одной и той же области исследуемого образца: А - первый вариант съёмки, Б - второй вариант съёмки, В - карта отличий кадров А и Б, полученная методом попиксельной разницы

Помимо малых локальных шумов и дефектов в процессе покадрового сканирования могут возникать крупномасштабные дефекты, дрейф характеристик. Например, на рис. 2.7 приведён пример нестабильности яркости и контраста в рамках одного кадра, приведён фрагмент уже совмещённого общего изображения.

На рис. 2.8 представлен пример перепада яркости и контраста в переделах строк кадров общего совмещённого изображения. Помимо цветовых характеристик так же может наблюдаться нестабильность фокусировки, астигматизм, нелинейные искажения на краях кадров.

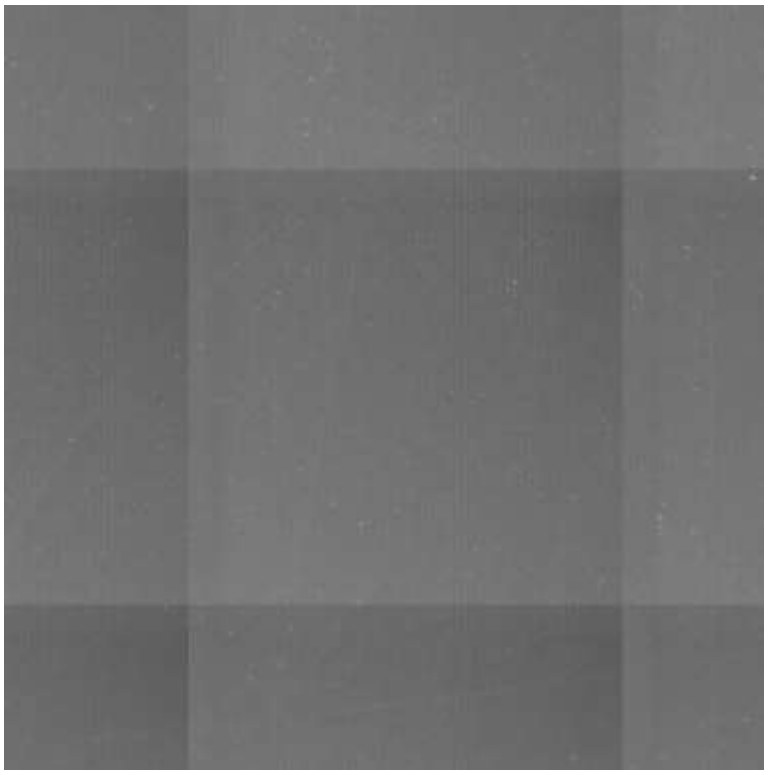


Рис. 2.7 Демонстрация нестабильности яркости и контраста в процессе сканирования отдельных кадров. Фрагмент общего совмещённого изображения

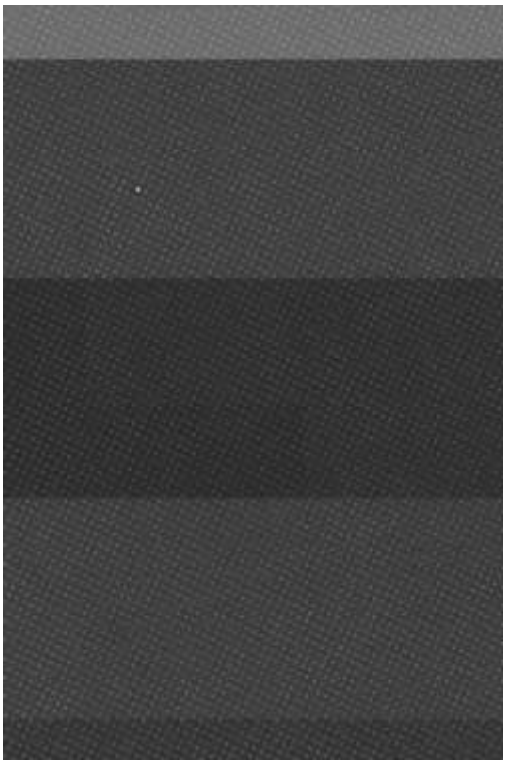


Рис. 2.8 Демонстрация нестабильности яркости и контраста при переходе от одного ряда кадров к другому. Фрагмент общего совмещённого изображения

Дополнительно ситуацию осложняют возможные спонтанные локальные дефекты различной величины, вызванные внешним воздействием на сканирующее оборудование. На рис. 2.9 приведён пример искажения изображения в результате подобных воздействий.

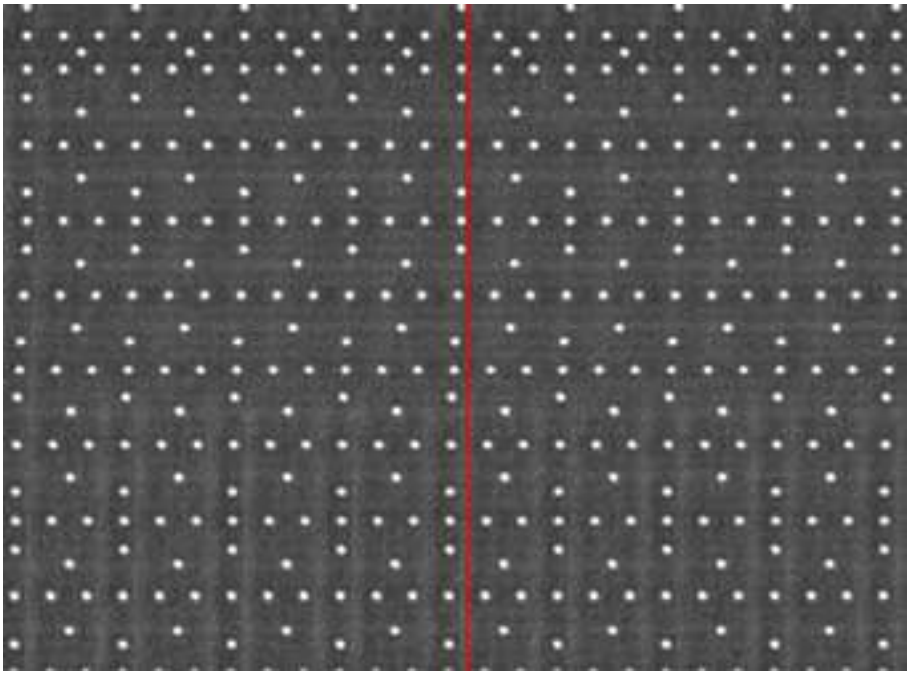


Рис. 2.9 Пример сбоя в процессе съёмки кадра, вызванного некоторым внешним воздействием. Красная линия является ориентиром, демонстрирующим искажение: белые точки, в действительности, располагаются строго симметрично, ортогонально

Дефекты могут возникать из-за различных электромагнитных помех, вибраций (в том числе и сейсмических), сбоев каких-либо систем оборудования и по иным причинам.

Подробное рассмотрение причин и механизмов возникновения подобных дефектов и шумов выходит за рамки данной диссертации, важен неотъемлемый факт их наличия на совмещаемых изображениях. Соответственно, как малые локальные, так и крупные дефекты оказывают влияние на процесс совмещения соседних кадров. Зачастую, степень зашумленности изображений, локальных отличий в перекрываемых областях соседних кадров настолько велика, что расчёт корректного относительного

положения двух соседних кадров методами прямого попиксельно сравнения не даёт корректных, однозначных результатов.

2.4 Методы решения

2.4.1 Известные методы совмещения

Для автоматизированного расчёта относительных положений для каждой пары соседних кадров при проведении плоской сшивки можно использовать следующие методики:

- расчёт относительных смещений кадров по первичным растровым данным **(А)**;
- расчёт относительных смещений кадров по предварительно обработанным растровым данным **(Б)**;
- расчёт относительных смещений кадров по данным, полученным в результате векторизации топологических объектов **(В)**.

У каждой из этих трёх методик есть свои сильные и слабые стороны:

А. Расчёт относительных смещений соседних кадров по первоначальным растровым данным является наиболее простым с методологической точки зрения, так как не требует никаких предварительных операций, оперирует с первичной информацией. При этом такой подход наименее устойчив перед разнообразием характеристик исходных. Во многих случаях характеристики шумов на изображениях одного порядка с характеристиками объектов интереса, соответственно в таких случаях без предварительной очистки от шума получить достоверный результат совмещения соседних кадров не представляется возможным.

Б. Предварительная обработка растровых данных с целью устранения шумов позволяет повысить в некоторой степени устойчивость процесса автоматизированного расчёта относительных смещений для пар кадров. Но по

причине большого разнообразия исходных изображений, во многих случаях к каждому конкретному результату съёмки придётся подбирать отдельный алгоритм устранения шумов для получения значительных результатов. Но чем выше степень обработки для устранения шумов, тем больше потенциальных реперных точек совмещения кадров теряется в области перекрытия.

В. Методика расчёта относительных смещений соседних кадров по распознанным, векторизованным данным наиболее устойчива перед шумами и малыми искажениями изображения. В то же время, в процессе векторизации теряется информация о мелких структурных особенностях в областях перекрытия кадров, которые могли бы быть использованы для точного совмещения пары соседних кадров, в области перекрытия которых отсутствуют векторные объекты (рис 2.10). Также потеря информации о мелких структурных особенностях значительно затрудняет совмещение кадров с регулярными симметричными топологическими объектами (рис. 2.11).

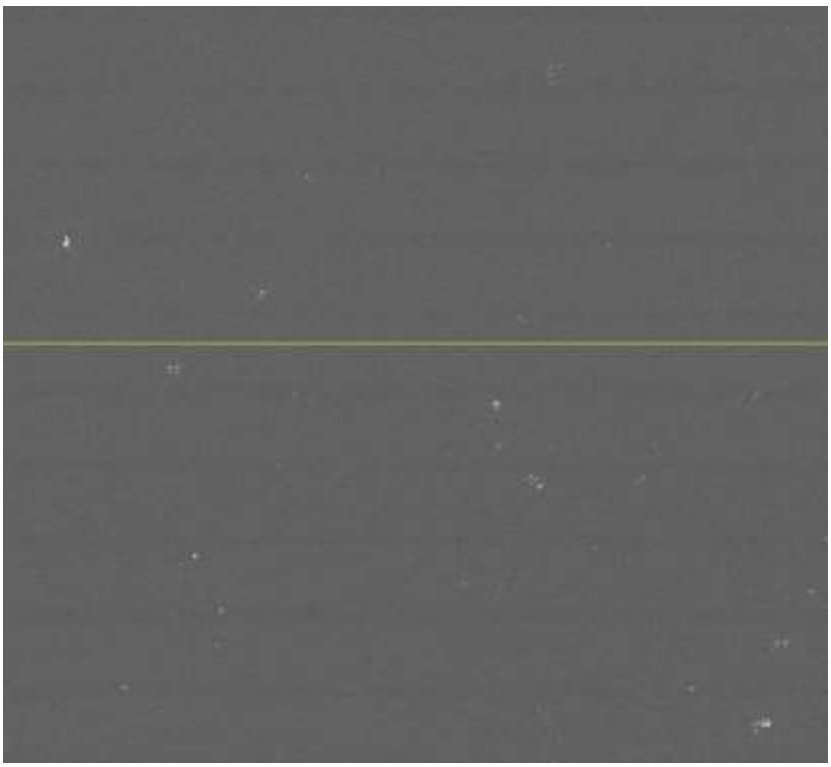


Рис. 2.10 Пример области перекрытия кадров, где есть мелкие дефекты, но нет векторных объектов. Жёлтая линия обозначает границу раздела двух соседних кадров

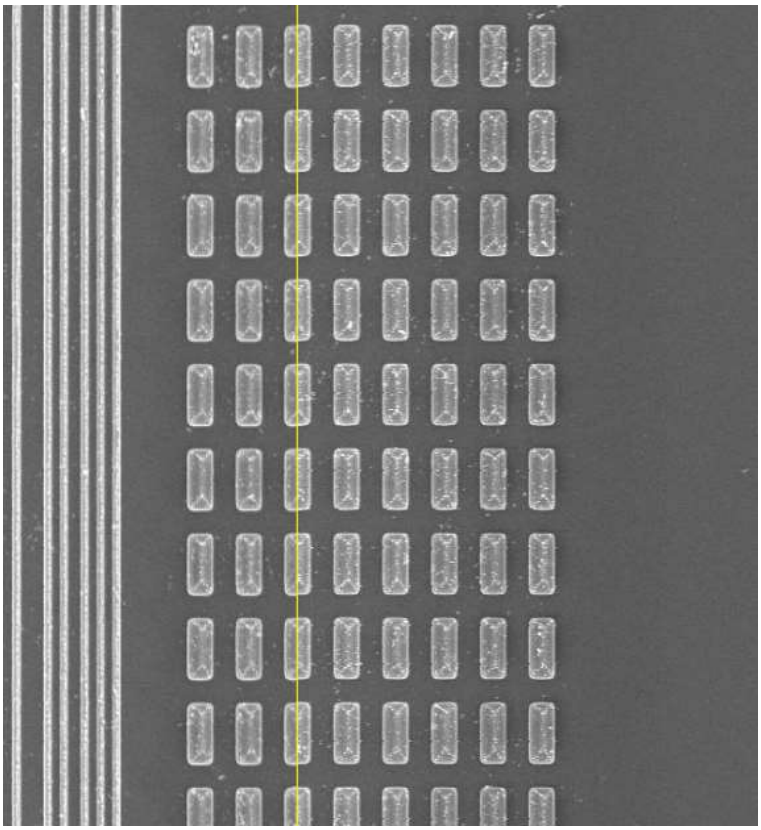


Рис. 2.11 Пример области перекрытия кадров с регулярными структурами. Жёлтая линия обозначает границу раздела двух соседних кадров

Расчёт относительных положений для пары соседних кадров при проведении плоской сшивки с использованием первичных растровых данных может быть реализован следующим образом:

1. Для каждого кадра из пары вырезается шовная область с запасом на полный диапазон смещений. Полный диапазон смещений определяет в каких пределах допустимо искать оптимальное положение одного кадра относительно другого (в подавляющем большинстве протестированных случаев достаточно ±15 пикселей).
2. Для каждого возможного взаимного положения шовных областей в пределах диапазона смещений рассчитывается метрика схожести D (2.1) двух изображений при наложении друг на друга. Пример изображений для

расчёта представлены на рис. 2.12. Результаты самого расчёта представлены в таблице 2.1.

3. Относительное смещение, при котором метрика схожести двух изображений показала наименьшее значение, устанавливается как результирующее относительное смещение кадров. Если значение метрики схожести указывает на несколько равнозначно лучших вариантов смещения, то за итоговый результат берётся то смещение, которое ближе к нулю. Если корректный расчёт произвести не удалось, либо если значение метрики схожести указывает на какое-либо крайнее положение смещения кадров, то в таком случае итоговый результат устанавливается как неопределённый.

Рис. 2.12 Визуализация связи метрики схожести двух изображений относительных положений кадров. Представлены RGB-совмещённые изображения области перекрытия кадров с различными вариантами относительных смещений. Результаты расчёта метрики и значения сдвигов представлены в таблице 2.1.

Таблица 2.1 Рассчет метрики схожести двух изображений (2.1) для изображений с рис. 2.12

Маркер изображения	**Относительный сдвиг по X**	**Относительный сдвиг по Y**	**Метрика схожести двух изображений D**
А	0	0	0.22
Б	-4	-5	8.24
В	10	9	13.87

Расчёт относительных положений для пары соседних кадров при проведении плоской сшивки с использованием предварительно обработанных растровых данных производится также, как и с использованием исходных растровых данных, за исключением того, что исходные изображения в первую очередь подвергаются какой-либо обработке с целью устранения шумов. Так как наиболее пагубное влияние на процесс расчёта относительного смещения оказывает мелкозернистый шум, размер которого составляет $\frac{1}{2}$ пикселя, то для устранения такого шума с одновременным сохранением наибольшего количества ключевых реперных точек на изображении может быть рекомендовано предварительное применение алгоритмов двустороннего размытия или медианного размытия [7] (рис. 2.13). Использование каких-либо более сложных, комплексных предварительных обработок изображений не демонстрирует должных преимуществ перед использованием векторных данных для расчёта смещений.

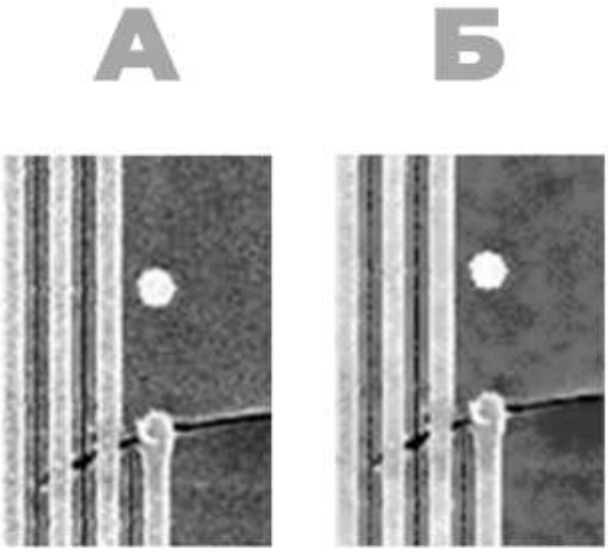


Рис. 2.13 Пример применения двустороннего размытия для устранения мелкозернистого шума: А - исходное изображение; Б - обработанное изображение

Расчёт относительных положений для пары соседних кадров при проведении плоской сшивки с использованием данных полученных в результате векторизации топологических объектов производится методом определения положения кадров с максимальной площадью взаимного перекрытия векторных объектов. Распознанные топологические объекты из шовной области первого кадра «накрываются» распознанными топологическими объектами второго кадра с учётом заданного смещения. Подсчитывается общая площадь перекрытых частей, что является смысловым эквивалентом метрики схожести двух изображений (2.1). Соответственно, относительное положение шовных областей двух кадров, при котором суммарное взаимное перекрытие векторных объектов максимально, устанавливается за итоговое смещение.

По сравнению с остальными подходами, расчёт относительных положений кадров по векторным данным продемонстрировал наиболее стабильные и точные результаты в случаях совмещения кадров, имеющих значительное количество мелких хорошо векторизованных объектов в области

перекрытия (рис. 2.14). И, напротив, для кадров в области перекрытия которых располагаются крупные монотонные объекты, либо векторизованные объекты отсутствуют, чрезвычайно затруднительно, или вовсе невозможно, получить точные результаты расчёта относительного положения пары кадров.

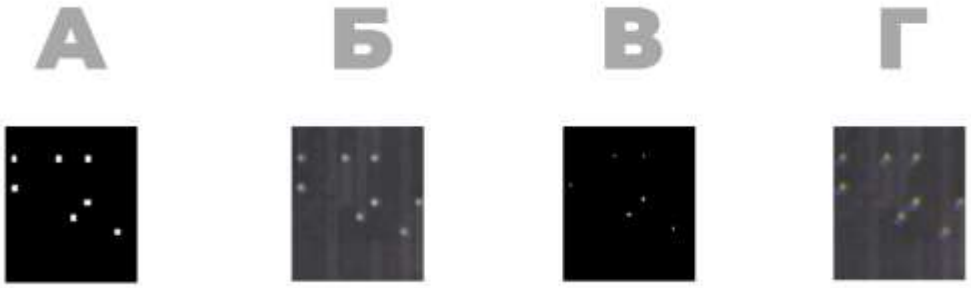


Рис. 2.14 Иллюстрация процесса расчёта взаимных смещений для пары кадров **А -** визуализация взаимных пересечений векторных объектов при наилучшем совмещении x = 0, y = 0; **Б -** RGB-совмещение при x = 0, y = 0; **В -** визуализация взаимных пересечений векторных объектов при x = 4, y = 4; **Г-** RGB-совмещение при x = 4, y = 4

Подходы с использованием векторных данных или с использованием растровых данных в некоторой степени компенсируют недостатки и преимущества друг друга. В зависимости от качества и характеристик исходных изображений использование одного из подходов может быть предпочтительнее. Но всё же при работе с изображениями реальных исследуемых объектов ни один из методов не позволяет получить точное совмещение абсолютно всех кадров. От нескольких единиц до нескольких десятков кадров на сотню как правило совмещается некорректно. Некорректные совмещения могут происходить по причине того, что в области перекрытия кадров отсутствуют какие-либо чёткие объекты, реперные точки, или изображение имеет монотонную регулярную структуру. Также значительный процент ошибочных совмещений может быть в случаях обработки изображений с высокой зашумленностью и плохо поддающихся векторизации. Некорректное совмещение может быть двух типов:

- в рамках алгоритма корректно совместить изображения не удалось;

- алгоритм отработал штатно, но при анализе общего пространства кадров очевидны аномалии (ошибки совмещения).

Если в первом случае некорректные совмещения изначально выявлены и могут быть легко интерпретированы человеком, то выявление ошибок совмещения требует отдельных действий и анализа. У всякого кадра есть четыре соседних кадра, если это кадр внутренней области слоя (без учёта угловых пересечений), и два или три соседних кадра для кадров по периметру общего изображения. Так как для каждой совмещаемой пары кадров взаимное смещение рассчитывается отдельно, то для каждого отдельного кадра слоя есть несколько отдельных результатов расчёта смещения. Сопоставление и анализ таких отдельных результатов между собой позволяет выявить ситуации, где расчёт попарного совмещения кадров был произведён с возможной ошибкой.

Положение первого кадра сшиваемого слоя (верхний левый кадр) в пространстве всего слоя берётся за точку начала отсчёта. Далее кадры подшиваются последовательно по одному, мозаично заполняя пространство общего изображения. Подшивание кадров может проводиться как сверху вниз, столбец за столбцом; так и слева на право, строка за строкой. Кадры первой строки последовательно подшиваются один к другому со смещением, рассчитанным между подшиваемым кадром и его левым соседом (уже добавленным в пространство слоя).

2.4.2 Предлагаемый метод

Исходя из описания вышеперечисленных методов, видно, что у них всех есть минусы. Поэтому была поставлена задача разработать более универсальный метод, который будет сочетать в себе плюсы уже известных методов, а также закроет хотя бы часть их недостатков.

Эти условия привели к поиску ответов в нейросетевом подходе. Нейросети уже зарекомендовали себя в подобного рода задачах (например, бинаризация объектов на изображении [2]), когда детерминированные методы дают промахи и имеют слабые моменты. Если собрать обширный и разнообразный датасет из входных данных, то нейросети могут показать результат не хуже, а может даже лучше, чем методы, построенные на строгих алгоритмах.

О самом нейросетевом подходе к решению рассматриваемой в данной диссертации задачи речь пойдет в разделе 2.5. Далее речь пойдет о сборе данных для обучения нейросети.

2.4.3 Сбор данных

В этом разделе будут рассмотрены методы и приемы, которые позволили собрать разнообразный набор входных данных и сформировать репрезентативные тренировочный и тестовый датасеты для обучения нейронной сети.

2.4.3.1 Метод нарезки изображений

Входными данными для решаемой задачи, сводя ее к совмещению двух кадров, являются два изображения. Но целиком эти два изображения не представляют интереса: необходимы только области перекрытия этих изображений. Как и в разделе 2.1. будут рассмотрены левое и правое изображения (для верхнего и нижнего процедура будет аналогичной с поправкой на размерности, связанной с положением кадров).

Отбрасывая части изображения, не входящие в области перекрытия, образуются два изображения одинаковой размерности: область перекрытия с левого изображения и область перекрытия с правого изображения.

Как уже было описано в разделе 2.3. исходные изображения имеют размерность $m \times n$ пикселей (чаще всего это квадратные изображения, в процессе экспериментов в основном использовались изображения размером 2000×2000, в качестве примера, который будет фигурировать далее была взята конкретная картинка размерами 1688×1688, чтобы избежать абстрактных фигур в изображениях и примеры были более ясными для читателя), а область перекрытия двух соседних кадров – s пикселей. Исходя из этих данных размер области перекрытия при совмещений двух горизонтально соседних кадров будет равен $m \times s$ пикселей (для двух вертикально соседних кадров $s \times n$ соответственно)

Но такой размер довольно громоздкий, чтобы подавать его на вход нейросети. Поэтому решено было свести задачу к еще более узкой: область перекрытия бьется на квадраты с размерами сторон равными размеру области перекрытия ($s \times s$). Для горизонтальных соседей разбиение происходит сверху вниз. Визуально разбиение можно наблюдать на рис. 2.15.

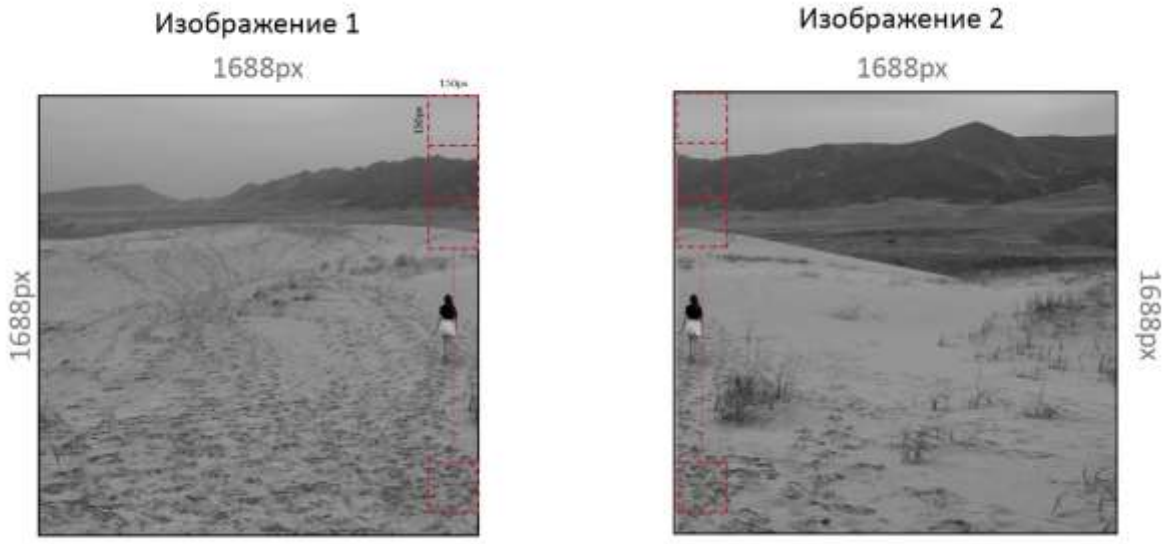


Рис. 2.15 Разбиение исходных изображений на квадраты

Тогда количество квадратов (sq) после разбиения можно вычислить по формуле:

$$sq = div(\frac{m}{s}), \tag{2.3}$$

где оператор *div* возвращает целую часть от деления.

Часть области перекрытия размером $red \times s$, где

$$red = \mathrm{mod}(\frac{m}{s}), \tag{2.4}$$

где mod – остаток от деления, отбрасывается и не участвует в дальнейших расчётах.

Соответственно, применяя данные преобразования на изображения из примера (рис. 2.15) область размером 1688×150 будет разбита на 11 квадратов размером 150×150 пикселей.

Теперь имеются два изображения из области перекрытия достаточно маленького размера, чтобы быть поданными на вход нейросети.

В следующем подразделе будет предложен метод совмещения двух изображений в градациях серого в одно RGB-изображение. Благодаря такому подходу на вход нейронной сети будет возможным подать одну матрицу размером $150 \times 150 \times 2$.

2.4.3.2 Метод совмещения изображений

Рассмотрим два изображения в градациях серого $imL = \begin{pmatrix} l_{11} & \mathrm{K} & l_{1s} \\ \mathrm{M} & \mathrm{O} & \mathrm{M} \\ l_{s1} & \mathrm{L} & l_{ss} \end{pmatrix}$ и $imR = \begin{pmatrix} r_{11} & \mathrm{K} & r_{1s} \\ \mathrm{M} & \mathrm{O} & \mathrm{M} \\ r_{s1} & \mathrm{L} & r_{ss} \end{pmatrix}$, которые представлены в виде матриц размером

Элементом этих матриц будет значение интенсивности пикселя от 0 до 255.

Далее предлагается метод совмещения двух изображений в одно. Важно заметить, что в данном случае под совмещением понимается не целевая задача данной работы, а лишь «упаковка» входных данных в более удобный для проектирования и тестирования формат.

Поскольку исходные изображения в градациях серого, то можно сказать, что размер матрицы равен $s \times s \times 1$. В то время как у RGB изображений - $s \times s \times 3$, где R = 1 – матрица красного канала, G = 2 – матрица зеленого канала, B = 3 – матрица синего канала.

Тогда для совмещения предлагается записать левое изображение *imL* в красный канал нового изображений (R), правое изображение *imR* в зеленый канал нового изображения (G), а в синий канал нового изображения (B) записать полсуммы каждого из элементов из *imL* и *imR* соответственно. Полученное изображение можно представить в виде матрицы:

$$imRGB = \begin{pmatrix} (l_{11}, r_{11}, \frac{1}{2}(l_{11} + r_{11}))_{11} & \text{K} & (l_{1s}, r_{1s}, \frac{1}{2}(l_{1s} + r_{1s}))_{1s} \\ \text{M} & \text{O} & \text{M} \\ (l_{s1}, r_{s1}, \frac{1}{2}(l_{s1} + r_{s1}))_{s1} & \text{L} & (l_{ss}, r_{ss}, \frac{1}{2}(l_{ss} + r_{ss}))_{ss} \end{pmatrix} \quad (2.5)$$

На рис. 2.16 можно наблюдать визуализацию процесса совмещения для отдельно взятого из итого изображения пикселя.

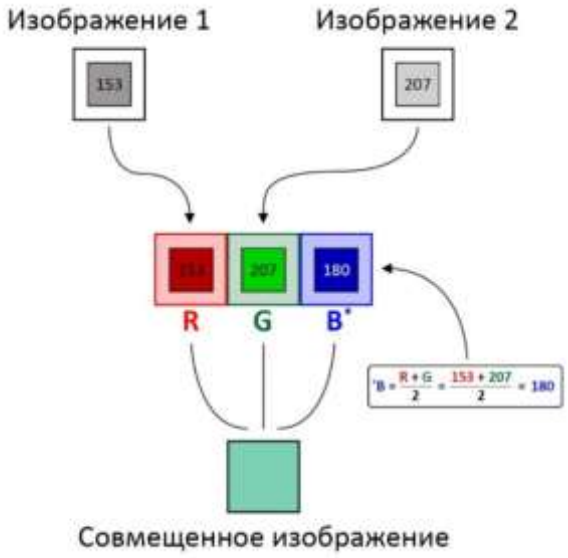


Рис. 2.16 Операция совмещения для отдельно взятого пикселя из изображения *imRGB*

На рис. 2.17 приведен пример совмещения двух изображений. Как можно видеть данный метод удобен не только «упаковкой» двух изображений в одно, но и также наглядно демонстрирует смещение одного изображения относительно другого. В данном примере правое изображение сдвинуто относительно левого на 4 пикселя X и 3 по Y. Это можно наблюдать по характерным сдвигам красного и зеленого каналов на результирующем изображении.

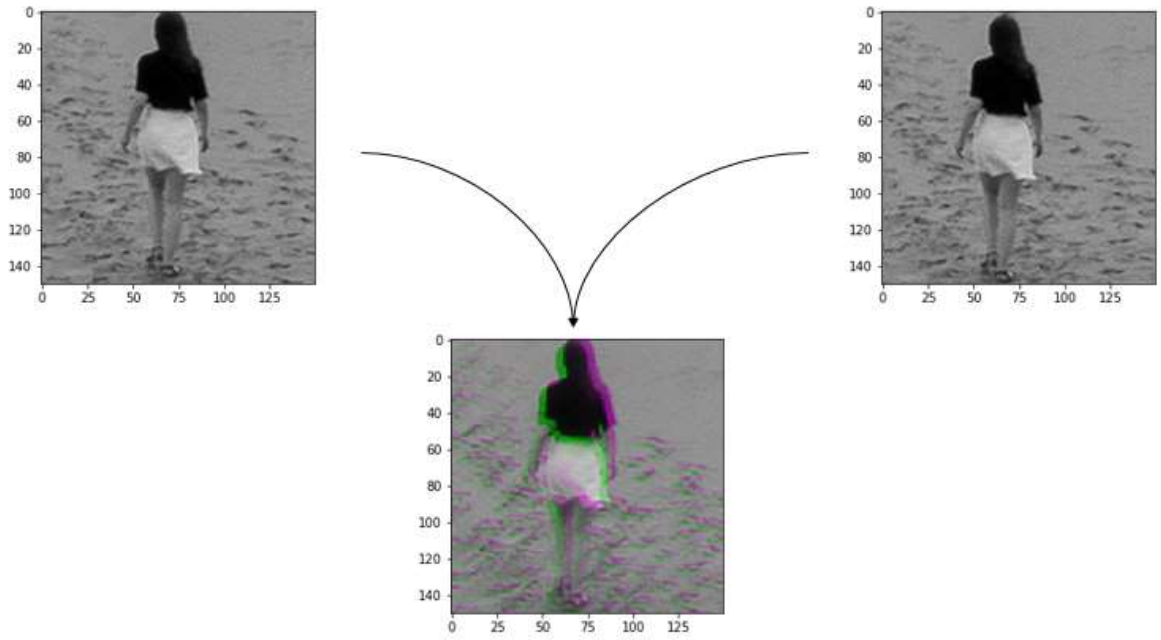


Рис. 2.17 Пример совмещения двух изображений в градациях серого в одно RGB-изображение

Вышеописанная процедура производится для всех вырезанных кусков из двух исходных смежных изображений. Каждый из них затем подается на вход нейросети и получает предсказание по сдвигу. Предсказанные сдвиги суммируются по осям и берется среднее с некоторыми условиями. Это среднее является предсказанием сдвига для двух смежных изображений. Подробнее о процедуре расчета итогового результата в следующих разделах.

2.4.3.3 Аугментация выборки

Решение почти любой задачи машинного обучение упирается в репрезентативность и разнообразность собранного датасета. Даже при условии больших объемов исходных изображений (у исследуемых объектов панорама может состоять из нескольких десятков тысяч кадров) не получится собрать довольно обширное разнообразие смещений кадров друг относительно друга. Из-за регулярных структур и однотонности исследуемых объектов, а также из-за того, что для соседних кадров из одного исследуемого образца чаще всего смещения не сильно различаются по значению, появилась необходимость в искусственной генерации датасета.

Для более точных результатов предсказания необходимо обучить нейронную сеть на как можно более широком и разнообразном датасете. Необходимо, чтобы были представлены все примеры смещений в диапазоне $[-\Delta s, \Delta s]$ по каждой из осей, где Δs - максимальное отклонение одного изображения относительно другого.

Для генерации репрезентативных данных несколько десятков исходных смежных изображений, представляющих наиболее высокое соответствие и разнообразие для целей задачи сшивки, были вручную совмещены. То есть Δs по X = 0, Y = 0. Конкретно для расчётов в данной диссертации Δs была приравнена к 10.

Далее исходные изображения итерационно сдвигались друг относительно друга и по X и по Y, и нарезались в куски 150 на 150 пикселей. Как это выглядит – визуально отображено на рис. 2.18.

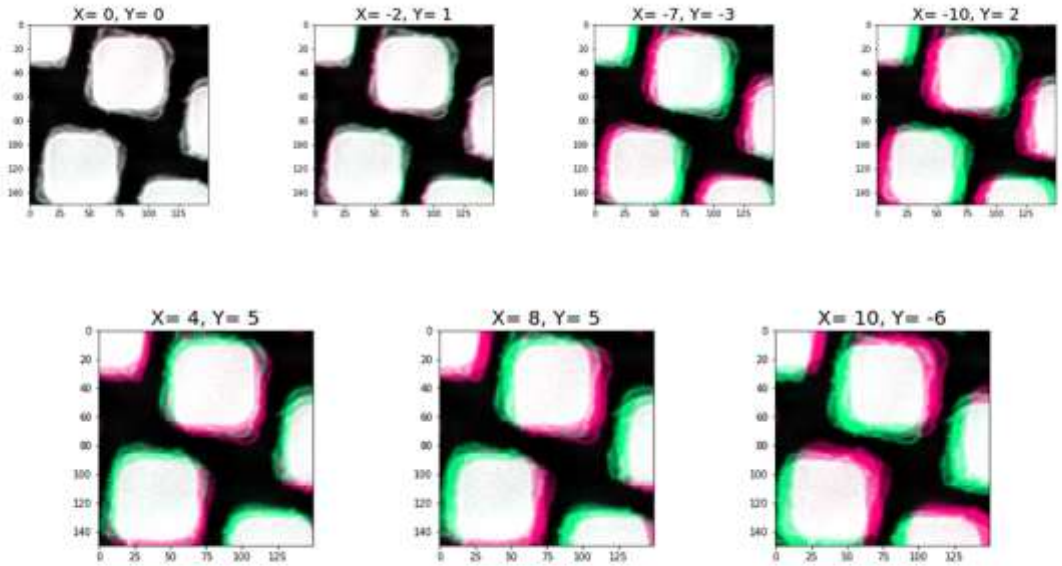


Рис. 2.18 Итерационное смещение изображений друг относительно друга, формирование датасета

Таким образом формировался датасет со всеми возможными смещениями в пределах $[-\Delta s, \Delta s]$ по каждой из осей.

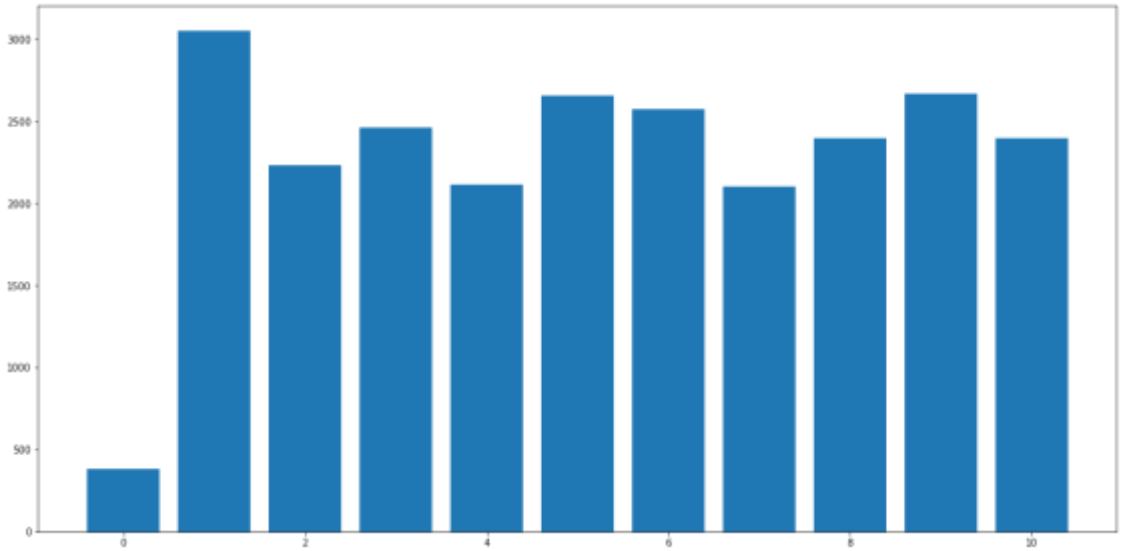


Рис. 2.19 Распределение максимальных значений сдвига (по оси X или Y) для всех изображений из сгенерированного датасета с $\Delta s = 10$

Чтобы датасет был более равномерно распределен, при его формировании применяется процедура просеивания и фильтрации данных по значению сдвигов. Таким образом для $\Delta s = 10$ итоговый сформированный датасет размером 30000 кадров имеет распределение по значениям максимального сдвига представленное на рис. 2.19.

В рамках работы над диссертацией в ходе экспериментов суммарно удалось собрать около 120000 изображений, но в формировании результирующей модели участвовал набор из 30000 с $\Delta s = 10$.

2.4.3.4 Первые четыре пикселя

Также стоит заметить, что генерация датасета производилась в прикладной программе, реализованной на языке C#, а обучение и тестирование модели производилось на языке Python. Из-за этого появилась необходимость связи данных (изображений) со значениями, которые ожидаются на выходе модели. Это также необходимо, чтобы данные из датасета были более «самостоятельными», то есть не хранили информацию о целевых переменных в каком-то сопровождающем файле. Хранить их в названии файла тоже не очень хорошая практика, ведь файл можно переименовать и тогда потеряются данные.

Поэтому было решено задействовать первый четыре пиксела синего канала изображения:

- первый – под знак значения сдвига по X;
- второй – под значение сдвига по X;
- третий - под знак значения сдвига по Y;
- четвертый - под значение сдвига по Y.

Знак определяется следующим образом – значение пикселя равное нулю эквивалентно минусу. Если же значение больше нуля, то плюс. Как это выглядит можно наблюдать на рис. 2.20.

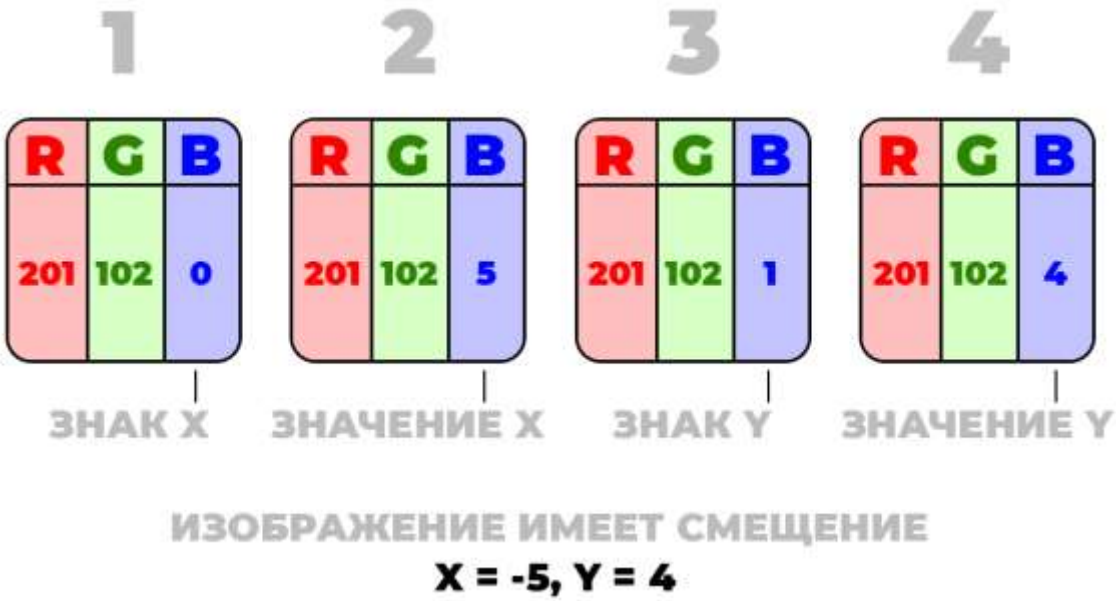


Рис. 2.20 Пример записи смещений в синий канал изображения

Таким образом при подготовке данных к обучению нейросети появилась возможность извлекать целевые переменные прямо из файлов изображений. Этот момент также не влияет на обучение, потому что при тренировке сети синий канал отбрасывается, на вход подается матрица $150 \times 150 \times 2$, хранящая информацию о двух совмещаемых изображениях.

2.5 Нейросетевой подход

За основу была взята сверточная нейронная сеть LeNet, предложенная Яном Лекуном в 1989 году [8]. Его модель была разработана для решения задачи классификации.

В начале экспериментов, проводимых в рамках данной диссертации, было проработан вариант сведения поставленной задачи к задаче классификации. Далее будет описан процесс эволюции модели в течение исследования.

2.5.1 Эволюция модели

На протяжении исследования модель прошла несколько этапов видоизменений и модернизаций с целью улучшения качества предсказаний. Как уже говорилось ранее на самых ранних этапах были попытки свести задачу к задаче классификации. Обозначим модель, которая была предложена для реализации, как нулевую.

2.5.1.1 Нулевая модель

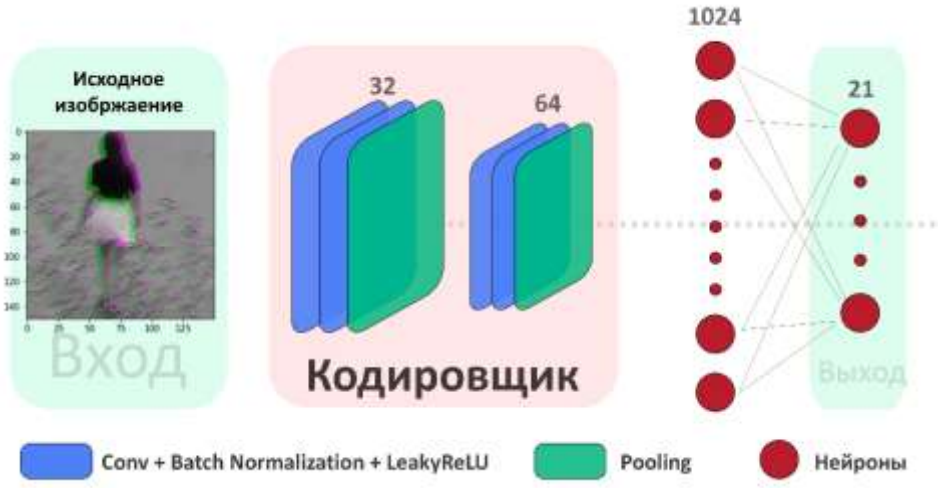


Рис. 2.21 Архитектура нулевой модели

Архитектуру предложенной модели можно наблюдать на рис. 2.21.

В задачах классификации один из выходных нейронов принимает значение 1, остальные 0. Каждый нейрон соответствует определенному классу, чей нейрон выдал единицу, к тому и относится классифицируемый входной образец. Предполагалось, что слой выходных нейронов с их количеством равным $\Delta s \times 2 + 1$ будет «зажигать» один из нейронов на выходе.

А его порядковый номер был бы значением смещения по одной из осей. Сеть пришлось бы обучать отдельно на X и на Y.

Данная модель показывала очень плохие результаты, к тому же из-за громоздкости реализации с ней было трудно работать. Поэтому было решено перейти к решению задачи регрессии.

2.5.1.2 Первая модель

Переход к задаче регрессии предполагает под собой два выходных нейрона, которые активируются линейной функцией. Они отвечают за значение сдвига по осям X и Y.

От нулевой модели был отброшен выходной слой и заменен на новый, состоящий из двух нейронов. Полученная модель уже показывала весьма неточные, но интерпретируемые результаты.

В процессе исследования над первой моделью были произведены модификации: а именно был добавлен пакет слоев свертки с количеством фильтров – 96.

В результате архитектуру итоговой первой модели можно наблюдать на рис. 2.22.

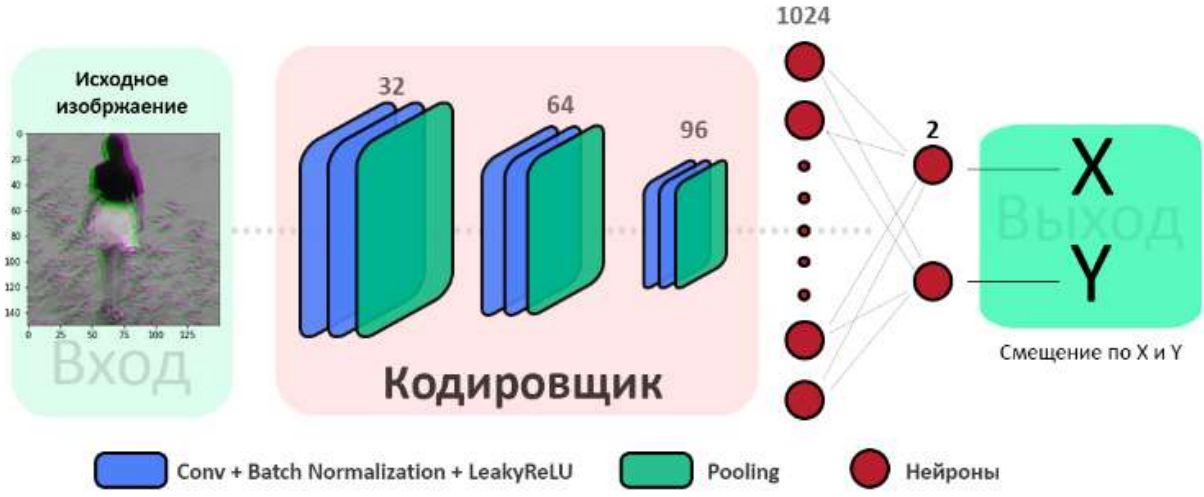


Рис. 2.22 Архитектору первой модели

Результаты стали еще лучше, можно наблюдать это по таблице 2.2.

2.5.1.3 Вторая модель

Для второй модели были взяты все слои от первой, добавлен пакет слоев свертки с количеством фильтров 128, а также полносвязный слой с количеством нейронов 512. Полученная сеть представлена на рис 2.23.

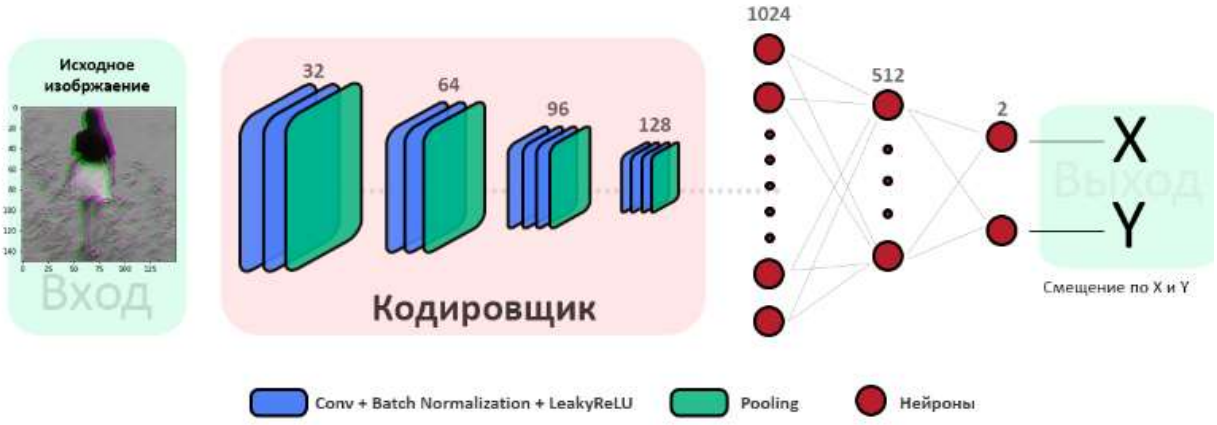


Рис. 2.23 Вторая модель нейронной сети

Вторая модель также подверглась модификациям и максимально приблизилась к итоговой модели: количество фильтров было увеличено 64, 96, 128, 160, для каждого пакета свертки соответственно.

Данная модель показала очень хорошие результаты (таблицы 2.2, 2.3). Но итоговой моделью стала нейронная сеть, описанная в подразделе ниже.

2.5.2 Итоговая модель нейронной сети

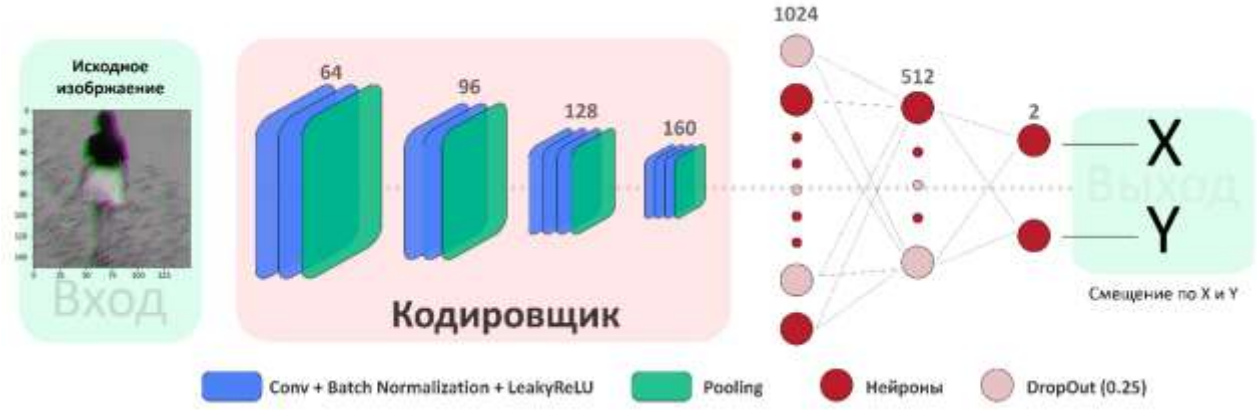


Рис. 2.24 Итоговая модель нейронной сети

Итоговая модель представляет собой следующую конструкцию: на вход нейронной сети подается изображение (матрица $150 \times 150 \times 2$), далее следует кодировщик, состоящий из четырех пакетов, которые включают в себя сверточные слои, слои активации и слои субдискретизации, где происходит уплотнение карт признаков. Количество фильтров у этих пакетов – 64, 96, 128, 160 соответственно. Далее идут два полносвязных слоя, на которых накладывается процедура dropout (отключение нейрона из слоя в процессе обучения с заданной вероятностью) со значением 0.25. И выходной слой состоящий из двух нейронов, которые отвечают за значение сдвигов по оси X и Y. Архитектура представлена на рис. 2.24

Результат работы всех описанных выше моделей (за исключением нулевой) приведен в сравнительных таблицах 2.2, 2.3.

Была сформирована выборка из n тестовых примеров. Пусть $X = (x_1, ..., x_n)$ - ожидаемые значения по сдвигам относительно оси X, $Y = (y_1, ..., y_n)$ - ожидаемые значения по сдвигам относительно оси Y, $\hat{X} = (\hat{x}_1, ..., \hat{x}_n)$ - значения по сдвигам относительно оси X предсказанные моделью, $\hat{Y} = (\hat{y}_1, ..., \hat{y}_n)$ - значения по сдвигам относительно оси Y предсказанные моделью.

Тогда задаются метрики, по которым проводится анализ:

- Среднее отклонение по максимальному значению X, Y:

$$\max Err = \frac{1}{n}\sum_{i=1}^{n}\max(|x_i - \hat{x}_i|, |y_i - \hat{y}_i|) \quad (2.6)$$

- Среднее отклонение по X:

$$xErr = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \quad (2.7)$$

- Среднее отклонение по Y:

$$yErr = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (2.8)$$

Таблица 2.2 Сравнительная таблица результатов работы моделей нейронной сети на тренировочном наборе данных

Название модели	**maxErr (2.6)**	**xErr (2.7)**	**yErr (2.8)**
Первая модель (два пакета свертки, один полносвязный слой)	14.304	6.802	12.292
Первая модель (три пакета свертки, один полносвязный слой)	8.041	5.865	5.87
Вторая модель (четыре пакета свертки, два полносвзяных слоя)	11.654	8.764	8.516
Вторая модель (четыре пакета свертки, два полносвзяных слоя, увеличенное число фильтров)	2.0	1.645	1.408
Итоговая модель	**1.879**	**1.501**	**1.389**

Таблица 2.3 Сравнительная таблица результатов работы моделей нейронной сети на валидационном наборе данных

Название модели	**maxErr (2.6)**	**xErr (2.7)**	**yErr (2.8)**
Первая модель (два пакета свертки, один полносвязный слой)	7.684	5.078	5.74
Первая модель (три пакета свертки, один полносвязный слой)	6.257	4.485	4.598
Вторая модель (четыре пакета свертки, два полносвзяных слоя)	7.684	6.255	5.009
Вторая модель (четыре пакета свертки, два полносвзяных слоя, увеличенное число фильтров)	2.439	1.882	1.483
Итоговая модель	**2.389**	**1.75**	**1.515**

Таким образом можно видеть, что поставленная задача о высокоточной сшивке решена. Отклонения по X и Y не превосходят двух пикселей в среднем.

2.5.3 Примеры работы нейронной сети

В этом подразделе будут приведены примеры работы итоговой нейронной сети на различных данных:

- таблица 2.4 – на целевых данных из тренировочного набора. Как можно наблюдать на данных из тренировочного набора сеть дает высокоточные предсказания на различных изображениях, в том числе с регулярной структурой;
- таблица 2.5 – на целевых данных из валидационного набора. На этом наборе модель ведет себя чуть, хуже, что ожидаемо, ведь это

валидационный набор данных, а значит сеть таких примеров раньше не видела. Но и тут результаты достаточно высоки. Стоит отметить третий пример. На картинке нет никаких объектов, чтобы взять за основу для совмещения. Если поставить задачу совмещения данного изображения человеку, то скорее всего он зайдет в тупик, либо даст неверные показатели. Однако нейросеть отлично справилась;

- таблица 2.6 – изображения созданы вручную в программе Photoshop. Данные кадры специально создавались, чтобы посмотреть спектр возможностей модели. На первом изображении – графический логотип хоккейной команды МАИ, на втором – пасмурное небо над морем, а на третьем – бегущая ящерица. Со всеми изображениями модель справилась, не превышая отклонения на 1 пиксель;
- таблица 2.7 – за основу взято третье изображение из таблицы 2.6. На изображение специально накладываются шумы и дефекты, чтобы оценить устойчивость сети к ним. На первом изображении у правой картинки была поднята яркость, на втором – снижена. Модель повела себя устойчиво и дала те же результаты, что и без примененных фильтров. На третьем, четвертом и пятом изображениях был наложен шум и дефекты. Модель смогла найти ответы и предсказать результат, не превышая отклонений на 1 пиксель.

Таблица 2.4 Результаты работы модели на целевых данных из тренировочного набора

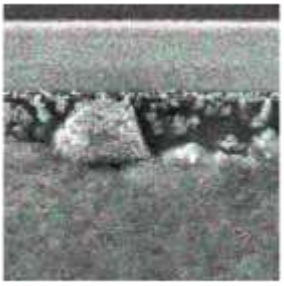
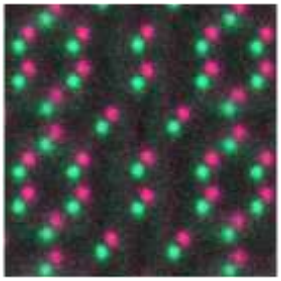
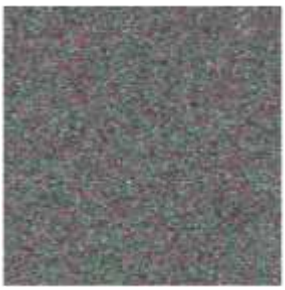
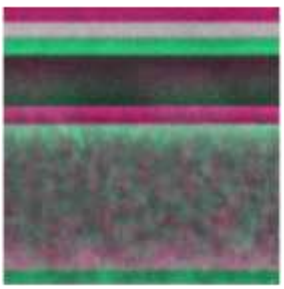
Входное изображение	**Ожидаемый результат** $(\Delta x, \Delta y)$	**Результат модели** $(\Delta x, \Delta y)$
	**(0, 0)**	Непрерывный **(0.00220987, -0.0925331)** Округленный до целых **(0, 0)**
	**(5, -7)**	Непрерывный **(5.108243, -6.8044996)** Округленный до целых **(5, -7)**
	**(3, 8)**	Непрерывный **(3.3179743, 7.84106)** Округленный до целых **(3, 8)**
	**(-1, 4)**	Непрерывный **(-1.876887, 3.5874705)** Округленный до целых **(-2, -4)**
	**(-10, -10)**	Непрерывный **(-9.466134, -9.195963)** Округленный до целых **(-9, -9)**

Таблица 2.5 Результаты работы модели на данных из валидационного набора

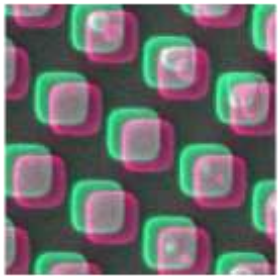
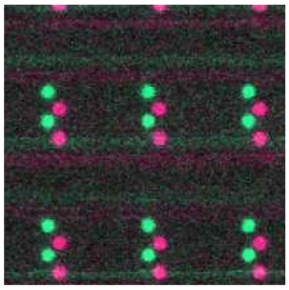
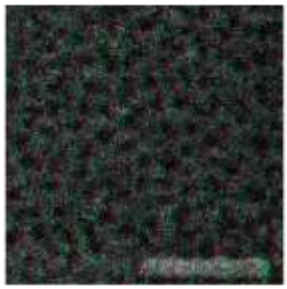
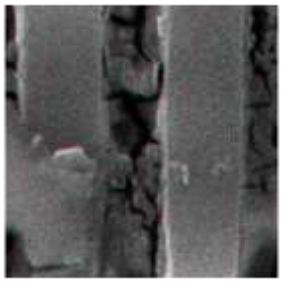
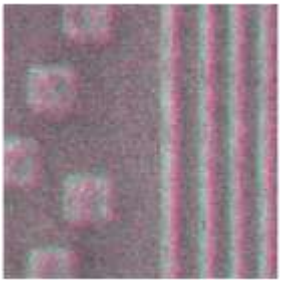
Входное изображение	**Ожидаемый результат** $(\Delta x, \Delta y)$	**Результат модели** $(\Delta x, \Delta y)$
	**(8, 6)**	Непрерывный **(9.114626, 5.2795663)** Округленный до целых **(9, 5)**
	**(6, 9)**	Непрерывный **(5.1071486, 8.833363)** Округленный до целых **(5, 9)**
	**(-1, 1)**	Непрерывный **(-2.2728744, -0.4127371)** Округленный до целых **(-2, 0)**
	**(0, -1)**	Непрерывный **(-0.22552773, -0.88309884)** Округленный до целых **(0, -1)**
	**(3, 3)**	Непрерывный **(3.364457, 3.3476079)** Округленный до целых **(3, 3)**

Таблица 2.6 Результаты работы модели на данных, собранных вручную

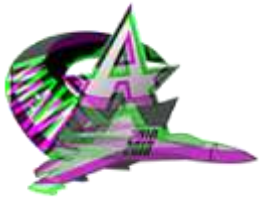
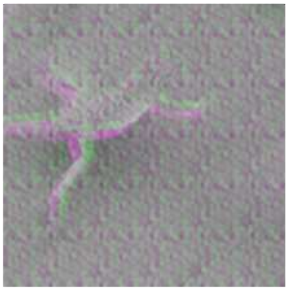
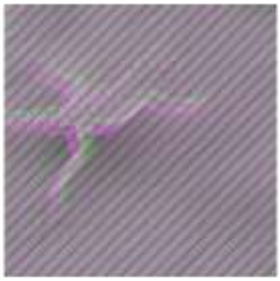
Входное изображение	**Ожидаемый результат** $(\Delta x, \Delta y)$	**Результат модели** $(\Delta x, \Delta y)$
	**(-3, 7)**	Непрерывный **(-3.4529185, 7.712231)** Округленный до целых **(-3, 8)**
	**(4, -6)**	Непрерывный **(4.556762, -6.2524137)** Округленный до целых **(5, -6)**
	**(-5, 5)**	Непрерывный **(-5.581642, 5.7536263)** Округленный до целых **(-6, 6)**

Таблица 2.7 Результаты работы модели на данных, собранных вручную с искусственными шумами

Входное изображение	**Ожидаемый результат** $(\Delta x, \Delta y)$	**Результат модели** $(\Delta x, \Delta y)$
	**(-5, 5)**	Непрерывный **(-4.32564, 5.065619)** Округленный до целых **(-4, 5)**

Продолжение таблицы 2.7

Входное изображение	**Ожидаемый результат** $(\Delta x, \Delta y)$	**Результат модели** $(\Delta x, \Delta y)$
	**(-5, 5)**	Непрерывный **(-4.349349, 4.658294)** Округленный до целых **(-4, 5)**
	**(-5, 5)**	Непрерывный **(-4.6734366, 5.4232736)** Округленный до целых **(-5, 5)**
	**(-5, 5)**	Непрерывный **(-4.81096, 5.4646997)** Округленный до целых **(-5, 5)**
	**(-5, 5)**	Непрерывный **(-5.627106, 6.1283083)** Округленный до целых **(-6, 6)**

Расчет результирующих значений смещений для двух смежных исходных кадров происходит по следующему алгоритму:

1. Рассчитанные значения смещений группируются в массивы по X и по Y;
2. Этим массивы сортируются, отбрасываются первые три значения и последние три значения. Такой ход помогает отсеять выбросы.

3. Элементы прореженных массивов суммируются и делятся на количество элементов в массиве. Таким образом получается итоговое смещение по X и Y для двух исходных смежных изображений.

Результат для примера с рис. 2.25 и 2.15:

- Ожидаемые значения: **X = -8, Y = 0;**
- Значения полученный с помощью нейросети **X = -8, Y = 1;**

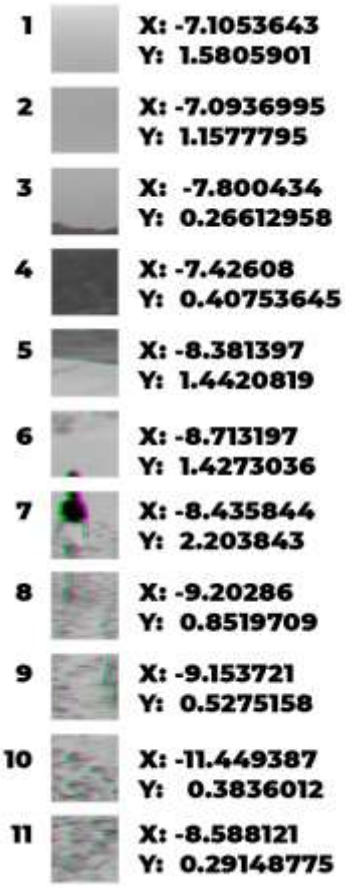


Рис. 2.25 Посчитанные смещения для нарезанных кусков области пересечения примера с рис. 2.15

Таким образом модель показывает свою работоспособность на приведенных примерах. Будь то примеры из тренировочных наборов данных, целевые изображения или изображения, созданные вручную для проверки.

2.6 Разработка программного обеспечения

Все вычисления производились на компьютере с процессором Intel I9-10900X, видеокартой Nvidia RTX GeForce 2080Ti, объем оперативной памяти вычислительного комплекса – 128 Гб.

В прикладной части данной магистерской диссертации использовались программные средства, приведенные в таблице 2.8.

Таблица 2.8 Используемые программные средства

Название	**Тип**	**Версия**
Python	Язык программирования	3.6.8
Keras	Библиотека	2.2.4
TensorFlow	Библиотека	1.13.1
NumPy	Библиотека	1.16.1
Scikit-learn	Библиотека	0.23.2
Pandas	Библиотека	1.1.3
Matplotlib	Библиотека	3.0.2
C#	Язык программирования	7.0
.Net Framework	Фреймворк	4.6.7
ipython (Jupyter Notebook)	Среда разработки	7.16.1
Visual Studio Community	Среда разработки	16.9.6
Visual Studio Code	Среда разработки	1.56.2

Реализация некоторых программных компонент (в том числе код итоговой модели) приведены в приложении 1. Листинг обучения модели в

приложении 2. В процессе разработки неоднократно совершалось обращение к источникам [9,10,11,12].

ЗАКЛЮЧЕНИЕ

В магистерской диссертации выполнены все поставленные задачи:

- разработана модель нейросети с архитектурой из 4 пакетов свертки, двух полносвязных слоев и двух выходных нейронов;
- обученная модель дает в среднем точность в 1-2 пикселя (максимальное отклонение по одной из осей);
- реализована прикладная программа, в которой осуществляется сбор датасета для обучения нейронной сети;
- собран датасет из более чем 100000 изображений;
- датасет генерируется на основе точно совмещённых человеком изображений;
- модель устойчива к шумам и дефектам.

Таким образом, потребность в высокоточной сшивке в прикладных задачах делает настоящую выпускную квалификационную работу актуальной.

Полученная модель имеет практическую значимость. Она может быть использована в прикладных задачах, например, для анализа панорамных снимков с микроскопа или аэросъемки. Модель поможет собрать цельную панораму с высокой точностью совмещения, что упростит анализ целевых объектов на панорамном изображении.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Sun, C., Müller, E., Meffert, M. et al. Analysis of crystal defects by scanning transmission electron microscopy (STEM) in a modern scanning electron microscope. Adv Struct Chem Imag 5, 1 (2019). https://doi.org/10.1186/s40679-019-0065-1
2. Quijada, R., Dura, R., Pallares, J. et al. Large-Area Automated Layout Extraction Methodology for Full-IC Reverse Engineering. J Hardw Syst Secur 2, 322–332 (2018). https://doi.org/10.1007/s41635-018-0051-4
3. Дудкин, А.А. Алгоритмы для восстановления топологии в задаче обратного проектирования интегральных схем // Вестник БрГТУ. Физика, математика, информатика., 2008. - № 5. - С. 47-52.
4. machinelearning.ru [Электронный ресурс]. URL: http://www.machinelearning.ru/ (Дата обращения: 05.10.2019)
5. Вьюгин В.В. «Математические основы теории машинного обучения и прогнозирования», М.: 2013. - 387 с.
6. Гудфеллоу Я., Бенджио И., Курвилль А., Глубокое обучение / пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил
7. Гонсалес Р., Вудс Р. Цифровая обработка изображений Издание 3-е, исправленное и дополненное Москва: Техносфера, 2012. – 1104 с. , ISBN 978-5-94836-331-8, 193 - 200с.
8. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). *Proceedings of the IEEE*. **86** (11): 2278–2324. doi:10.1109/5.726791.
9. Документация библиотеки TensorFlow [Электронный ресурс]. URL: https://www.tensorflow.org/api_docs (дата обращения 10.12.2020)
10. Документация библиотеки NumPy [Электронный ресурс]. URL: https://numpy.org/doc/ (дата обращения 15.12.2020)
11. Документация языка C# [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (Дата обращения: 11.01.2021)

12. Траск Эндрю, Грокаем глубокое обучение. — СПб.: Питер, 2019. — 352 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-1334-7
13. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65 (6): 386--408* (*1958*)
14. NeuroHive [Электронный ресурс] URL: https://neurohive.io/ru/ (Дата обращения: 16.01.2021)

ПРИЛОЖЕНИЯ

Приложение 1

Основная часть программного кода на языке Python

Библиотека для предварительной обработки данных

- nnpreprocessing.py

```
import os
import numpy as np
from PIL import Image

from matplotlib import image
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import tqdm

def get_deltas(image):
    """Получает значения дельт (смещение по X и Y), которые хранятся в первых
    четырех пикселях синего канала изображения\n
        0 - знак X;
        1 - значение X;
        2 - знак Y;
        3 - значение Y

    Parameters
    ----------
    image: трехмерный массив изображение с размерностью (n, m, 3), где \n
        n - ширина изображения;
        m - высота изображения;
        3 - количество каналов (R, G, B)

    Returns
    -------
    deltas : array
        массив из двух челочисленных значений - сдвиг по X и сдвиг по Y

    """
    sign_of_dx = (0, 0, 2)
    value_of_dx = (0, 1, 2)

    sign_of_dy = (0, 2, 2)
    value_of_dy = (0, 3, 2)

    deltaX = (1 if image[sign_of_dx] else  -1) * image[value_of_dx]
    deltaY = (1 if image[sign_of_dy] else  -1) * image[value_of_dy]

    return np.array([deltaX, deltaY])
```

```
def get_images_with_deltas_from_dir(path, extension, max_delta=50, channels=2):
    """Получение всех картинок с указанным количеством каналов и вычеслинными дельтам
и из указнной директории с указанным расширением.
    Также происходит номировка значений в диапазон [0; 1] для последующего использова
ния нейронными сетями.


    Parameters
    ----------
    path: путь, из которого будут взяты все лежащие там файлы
    extension: расширение, по которому будут отбираться файлы
    channels: количество каналов, которые будут записаны в выходной массив

    Returns
    -------
    tuple: кортеж из двух массивов. \n
        Первый элемент - массив картинок с обрезанным синим каналом с размерностью (l
, n, m, channels),
        где\n
            l - количество отобранных изображений с указанным расширением;
            n - ширина изображений;
            m - высота изображений;
            channels - R, RG, RGB ~ (1, 2, 3).\n
        Второй элемент - массив дельт с размерностью (l, 2), где - 2 - смещение по X
и по Y соотвественно

    """
    files_in_path = os.listdir(path)

    im_arrs = []
    deltas = []

    for file in tqdm.tqdm_notebook(files_in_path):
        if not file.endswith(extension):
            continue

        image = Image.open(path + file)
        RGB_array = np.array(image)

        d = get_deltas(RGB_array)

        if abs(d[0]) > max_delta or abs(d[1]) > max_delta:
            continue

        RG_array = np.array(RGB_array[:, :, :channels]) #обрезаются каналы
        RG_array_norm = np.true_divide(RG_array, 255) #нормировка для НС (значения в
диапазоне [0, 1])

        im_arrs.append(RG_array_norm)
        deltas.append(d)
```

```
    return np.array(im_arrs), np.array(deltas)


def do_delta_vector(deltas, max_delta):
    """Перевод значения дельт в категориальный вектор. Т.е. вектор, в котором одно зн
ачение равно 1, все остальные 0.

    Parameters
    ----------
    deltas: массив смещений по (X, Y)
    max_delta: максимальное смещение, для формирования вектора

    Returns
    -------
    array: массив из пар векторов размерностью (max_delta + 1) - [X_n, Y_n]
    """

    zero = max_delta + 1
    size = zero + max_delta
    new_deltas = []
    for el in deltas:
        dx = np.zeros(size)
        dy = np.zeros(size)

        dx[zero + el[0] - 1] = 1
        dy[zero + el[1] - 1] = 1

        new_deltas.append((dx, dy))

    return np.array(new_deltas)


def seive(image_deltas, delta):
    """Фильтрация изображений в зависимости от значения сдвига.

    Parameters
    ----------
    image_deltas: массив изображений и дельт. image_deltas[0] - изображения, image_de
ltas[1] - дельты
    delta: предельной значние дельты, все что больше - отсеиваются

    Returns
    -------
    array: новый отфильтрованный массив изображений и дельт
    """

    imgs = image_deltas[0]
    deltas = image_deltas[1]
    new_imgs = []
```

```
    new_deltas =  []

    for i, cur_deltas in enumerate(deltas):

        if abs(cur_deltas[0]) > delta or abs(cur_deltas[1]) > delta:
            continue

        new_imgs.append(imgs[i])
        new_deltas.append(deltas[i])

    return np.array(new_imgs), np.array(new_deltas)

from shutil import copyfile

def copy_to_folder_less_then_delta_like(path, new_path, max_d, ext):
    files_in_path = os.listdir(path)

    for file in tqdm.tqdm_notebook(files_in_path):
        if not file.endswith(ext):
            continue

        image = Image.open(path + file)
        RGB_array = np.array(image)

        d = get_deltas(RGB_array)

        if abs(d[0]) <= max_d and abs(d[1]) <= max_d:
            copyfile(path + file, new_path + file)
```

Код итоговой модели:

```
def BDSM_net_4pack_morefilts_dropout():
    input_shape = (150, 150, 2)
    vc1 = 64
    vc2 = 96
    vc3 = 128
    vc4 = 160

    INIT = initializers.he_uniform()

    model = Sequential()

    model.add(Conv2D(vc1, kernel_size=3, padding="same", input_shape=input_shape, ker
nel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc1, kernel_size=3, padding="same", kernel_initializer = INIT))
```

```
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2, padding="same"))

    model.add(Conv2D(vc2, kernel_size=3, padding="same", input_shape=input_shape, ker
nel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc2, kernel_size=3, padding="same", kernel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2, padding="same"))

    model.add(Conv2D(vc3, kernel_size=3, padding="same", input_shape=input_shape, ker
nel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc3, kernel_size=3, padding="same", kernel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc3, kernel_size=3, padding="same", kernel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2, padding="same"))

    model.add(Conv2D(vc4, kernel_size=3, padding="same", input_shape=input_shape, ker
nel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc4, kernel_size=3, padding="same", kernel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(vc4, kernel_size=3, padding="same", kernel_initializer = INIT))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=2, padding="same"))

    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.25))

    model.add(Dense(2, activation='linear'))

    return model
```

Приложение 2

Пример обучения модели

1. Параметры компиляции

```
bdsm_net = nets.BDSM_net_4pack_morefilts_dropout()
bdsm_net.summary()
mc =
ModelCheckpoint("bdsm_4pack_2dense_less10_morefilters_correctRGB_Desoo
Data.h5",monitor="val_loss", verbose=1, save_best_only=True,
mode="auto")
red_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3,
min_delta=0.1, mode='min', verbose=1, min_lr=1e-8)

bdsm_net.compile(optimizer='adam', loss='mse', metrics=['mse',
'accuracy'])
hist_bdsm_net = bdsm_net.fit(X_train, y_train, batch_size=16,
epochs=30, verbose=2, validation_data=(X_test, y_test), callbacks=[mc,
red_lr])
```

2. Процесс обучения

```
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 150, 150, 64)      1216
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 150, 150, 64)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 150, 150, 64)      256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 150, 150, 64)      36928
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 150, 150, 64)      0
_________________________________________________________________
batch_normalization_2 (Batch (None, 150, 150, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 75, 75, 96)        55392
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 75, 75, 96)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 75, 75, 96)        384
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 75, 75, 96)        83040
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 75, 75, 96)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 75, 75, 96)        384
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 38, 38, 96)        0
_________________________________________________________________
```

```
conv2d_5 (Conv2D)            (None, 38, 38, 128)       110720
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 38, 38, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 38, 38, 128)       512
conv2d_6 (Conv2D)            (None, 38, 38, 128)       147584
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 38, 38, 128)       0
_________________________________________________________________
batch_normalization_6 (Batch (None, 38, 38, 128)       512
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 38, 38, 128)       147584
leaky_re_lu_7 (LeakyReLU)    (None, 38, 38, 128)       0
_________________________________________________________________
batch_normalization_7 (Batch (None, 38, 38, 128)       512
max_pooling2d_3 (MaxPooling2 (None, 19, 19, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 19, 19, 160)       184480
leaky_re_lu_8 (LeakyReLU)    (None, 19, 19, 160)       0
_________________________________________________________________
batch_normalization_8 (Batch (None, 19, 19, 160)       640
conv2d_9 (Conv2D)            (None, 19, 19, 160)       230560
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 19, 19, 160)       0
_________________________________________________________________
batch_normalization_9 (Batch (None, 19, 19, 160)       640
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 19, 19, 160)       230560
leaky_re_lu_10 (LeakyReLU)   (None, 19, 19, 160)       0
_________________________________________________________________
batch_normalization_10 (Batc (None, 19, 19, 160)       640
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 160)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              16385024
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
dense_3 (Dense)              (None, 2)                 1026
=================================================================
Total params: 18,143,650
Trainable params: 18,141,282
Non-trainable params: 2,368
_________________________________________________________________
```

```
WARNING:tensorflow:From c:\program files\python36\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use tf.cast instead.
Train on 16734 samples, validate on 8243 samples
Epoch 1/30
 - 84s - loss: 21.8295 - mean_squared_error: 3.4218 - acc: 0.7533 -
val_loss: 9.0864 - val_mean_squared_error: 2.1929 - val_acc: 0.8729

Epoch 00001: val_loss improved from inf to 9.08640, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 2/30
 - 78s - loss: 7.0645 - mean_squared_error: 1.9055 - acc: 0.8795 -
val_loss: 3.6872 - val_mean_squared_error: 1.3271 - val_acc: 0.9220

Epoch 00002: val_loss improved from 9.08640 to 3.68722, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 3/30
 - 80s - loss: 4.0383 - mean_squared_error: 1.4269 - acc: 0.9131 -
val_loss: 3.2363 - val_mean_squared_error: 1.2527 - val_acc: 0.9267

Epoch 00003: val_loss improved from 3.68722 to 3.23630, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 4/30
 - 79s - loss: 2.6857 - mean_squared_error: 1.1803 - acc: 0.9301 -
val_loss: 2.5688 - val_mean_squared_error: 1.1403 - val_acc: 0.9157

Epoch 00004: val_loss improved from 3.23630 to 2.56883, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 5/30
 - 78s - loss: 2.4539 - mean_squared_error: 1.1308 - acc: 0.9282 -
val_loss: 2.5722 - val_mean_squared_error: 1.1675 - val_acc: 0.9385

Epoch 00005: val_loss did not improve from 2.56883
Epoch 6/30
 - 78s - loss: 1.8446 - mean_squared_error: 1.0030 - acc: 0.9431 -
val_loss: 1.8015 - val_mean_squared_error: 0.9559 - val_acc: 0.9392
```

```
Epoch 00006: val_loss improved from 2.56883 to 1.80151, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 7/30
 - 79s - loss: 1.6394 - mean_squared_error: 0.9527 - acc: 0.9473 -
val_loss: 1.3536 - val_mean_squared_error: 0.8504 - val_acc: 0.9402
Epoch 00007: val_loss improved from 1.80151 to 1.35358, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 8/30
 - 78s - loss: 1.4229 - mean_squared_error: 0.8975 - acc: 0.9510 -
val_loss: 1.2667 - val_mean_squared_error: 0.7992 - val_acc: 0.9557

Epoch 00008: val_loss improved from 1.35358 to 1.26667, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 9/30
 - 75s - loss: 1.1983 - mean_squared_error: 0.8219 - acc: 0.9545 -
val_loss: 1.2506 - val_mean_squared_error: 0.7989 - val_acc: 0.9507



Epoch 00009: val_loss improved from 1.26667 to 1.25063, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 10/30
 - 75s - loss: 1.1776 - mean_squared_error: 0.8141 - acc: 0.9534 -
val_loss: 3.4008 - val_mean_squared_error: 1.2732 - val_acc: 0.9376

Epoch 00010: val_loss did not improve from 1.25063
Epoch 11/30
 - 79s - loss: 1.1604 - mean_squared_error: 0.8058 - acc: 0.9529 -
val_loss: 1.1565 - val_mean_squared_error: 0.8256 - val_acc: 0.9520

Epoch 00011: val_loss improved from 1.25063 to 1.15653, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 12/30
 - 78s - loss: 1.0352 - mean_squared_error: 0.7507 - acc: 0.9566 -
val_loss: 0.8247 - val_mean_squared_error: 0.6488 - val_acc: 0.9583

Epoch 00012: val_loss improved from 1.15653 to 0.82466, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 13/30
```

```
 - 79s - loss: 0.7626 - mean_squared_error: 0.6623 - acc: 0.9622 -
val_loss: 0.6403 - val_mean_squared_error: 0.5651 - val_acc: 0.9612

Epoch 00013: val_loss improved from 0.82466 to 0.64032, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 14/30
 - 78s - loss: 0.7154 - mean_squared_error: 0.6431 - acc: 0.9637 -
val_loss: 0.6279 - val_mean_squared_error: 0.5626 - val_acc: 0.9694

Epoch 00014: val_loss improved from 0.64032 to 0.62789, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 15/30
 - 79s - loss: 0.6321 - mean_squared_error: 0.6046 - acc: 0.9659 -
val_loss: 0.8759 - val_mean_squared_error: 0.7087 - val_acc: 0.9632

Epoch 00015: val_loss did not improve from 0.62789
Epoch 16/30
 - 80s - loss: 0.6083 - mean_squared_error: 0.5889 - acc: 0.9647 -
val_loss: 0.7475 - val_mean_squared_error: 0.6081 - val_acc: 0.9677

Epoch 00016: val_loss did not improve from 0.62789

Epoch 00016: ReduceLROnPlateau reducing learning rate to
0.00010000000474974513.
Epoch 17/30
 - 79s - loss: 0.3263 - mean_squared_error: 0.4256 - acc: 0.9711 -
val_loss: 0.2808 - val_mean_squared_error: 0.3591 - val_acc: 0.9760

Epoch 00017: val_loss improved from 0.62789 to 0.28076, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 18/30
 - 78s - loss: 0.2139 - mean_squared_error: 0.3541 - acc: 0.9750 -
val_loss: 0.2858 - val_mean_squared_error: 0.3695 - val_acc: 0.9712

Epoch 00018: val_loss did not improve from 0.28076
Epoch 19/30
 - 79s - loss: 0.1792 - mean_squared_error: 0.3277 - acc: 0.9753 -
val_loss: 0.2253 - val_mean_squared_error: 0.3233 - val_acc: 0.9742
```

```
Epoch 00019: val_loss improved from 0.28076 to 0.22532, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 20/30
 - 80s - loss: 0.1566 - mean_squared_error: 0.3062 - acc: 0.9781 -
val_loss: 0.2287 - val_mean_squared_error: 0.3227 - val_acc: 0.9767

Epoch 00020: val_loss did not improve from 0.22532

Epoch 00020: ReduceLROnPlateau reducing learning rate to
1.0000000474974514e-05.
Epoch 21/30
 - 78s - loss: 0.1273 - mean_squared_error: 0.2773 - acc: 0.9782 -
val_loss: 0.1951 - val_mean_squared_error: 0.2891 - val_acc: 0.9772

Epoch 00021: val_loss improved from 0.22532 to 0.19513, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 22/30
 - 79s - loss: 0.1238 - mean_squared_error: 0.2734 - acc: 0.9779 -
val_loss: 0.1957 - val_mean_squared_error: 0.2873 - val_acc: 0.9756

Epoch 00022: val_loss did not improve from 0.19513
Epoch 23/30
 - 79s - loss: 0.1179 - mean_squared_error: 0.2670 - acc: 0.9771 -
val_loss: 0.1938 - val_mean_squared_error: 0.2859 - val_acc: 0.9765

Epoch 00023: val_loss improved from 0.19513 to 0.19379, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5

Epoch 00023: ReduceLROnPlateau reducing learning rate to
1.0000000656873453e-06.
Epoch 24/30
 - 79s - loss: 0.1160 - mean_squared_error: 0.2639 - acc: 0.9781 -
val_loss: 0.1915 - val_mean_squared_error: 0.2844 - val_acc: 0.9763

Epoch 00024: val_loss improved from 0.19379 to 0.19154, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5
Epoch 25/30
```

```
 - 79s - loss: 0.1166 - mean_squared_error: 0.2657 - acc: 0.9782 -
val_loss: 0.1921 - val_mean_squared_error: 0.2845 - val_acc: 0.9754

Epoch 00025: val_loss did not improve from 0.19154
Epoch 26/30
 - 77s - loss: 0.1164 - mean_squared_error: 0.2645 - acc: 0.9789 -
val_loss: 0.1904 - val_mean_squared_error: 0.2833 - val_acc: 0.9756

Epoch 00026: val_loss improved from 0.19154 to 0.19039, saving model to
bdsm_4pack_2dense_less10_morefilters_correctRGB_BD.h5

Epoch 00026: ReduceLROnPlateau reducing learning rate to
1.0000001111620805e-07.
Epoch 27/30
 - 74s - loss: 0.1154 - mean_squared_error: 0.2642 - acc: 0.9771 -
val_loss: 0.1910 - val_mean_squared_error: 0.2840 - val_acc: 0.9763

Epoch 00027: val_loss did not improve from 0.19039
Epoch 28/30
 - 79s - loss: 0.1167 - mean_squared_error: 0.2651 - acc: 0.9779 -
val_loss: 0.1907 - val_mean_squared_error: 0.2837 - val_acc: 0.9754

Epoch 00028: val_loss did not improve from 0.19039
Epoch 29/30
 - 79s - loss: 0.1171 - mean_squared_error: 0.2651 - acc: 0.9763 -
val_loss: 0.1908 - val_mean_squared_error: 0.2841 - val_acc: 0.9761

Epoch 00029: val_loss did not improve from 0.19039

Epoch 00029: ReduceLROnPlateau reducing learning rate to
1.000000082740371e-08.
Epoch 30/30
 - 79s - loss: 0.1150 - mean_squared_error: 0.2641 - acc: 0.9782 -
val_loss: 0.1911 - val_mean_squared_error: 0.2837 - val_acc: 0.9762

Epoch 00030: val_loss did not improve from 0.19039
```