МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра Веб-технологий и компьютерного моделирования

ЖУК
Алесь Олегович

РАСПОЗНАВАНИЕ ЗДАНИЙ НА СПУТНИКОВЫХ СНИМКАХ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

Магистерская диссертация

Научный руководитель:
доктор технических наук, профессор
С.В. Абламейко

Допущена к защите

«____» _____ 2018 г.
Заведующий кафедрой
веб технологий и компьютерного моделирования
кандидат физико-математических наук,
доцент В.С. Романчик

Минск 2018

Содержание

ВВЕДЕНИЕ

Автоматическое распознавание зданий на спутниковых изображениях является важной и достаточно трудной задачей. Выделение зданий на спутниковых снимках может помочь в построении карт местности, определении численности людей, созданию плана застройки, нахождении вредоносных и незаконно построенных объектов. Сложность задачи заключается в большом числе различных конструкций у зданий.

В данной работе использовался большой объём тренировочных данных. Которые представляют собой снимки городов и маски этих городов с выделенными зданиями. Для тестирования и проверки результатов использовались снимки других городов. Задачей являлось оценить возможность обобщения алгоритма выделения зданий на снимках.

В данной работе продемонстрирован подход, с использованием глубоких нейронных сетей, для выделения и сегментации зданий на изображениях.

РЭФЕРАТ

Магістарская праца змяшчае: 44 старонкі, 22 іллюстрацыі, 1 прыкладанне, 16 выкарыстанных крыніц.

Ключавыя словы: глыбокія нейронныя сеткі, распазнанне бдынкаў, сегментацыя, градыентны спуск, функцыя кошту, функцыя актывацыі, архітэктура сеткі.

Аб'ектам даследавання з’яўляецца выкарыстанне нейронных сетак для выяўлення будынкаў на здымках зямной паверхні.

Мэтай магістарскай дысертацыі з'яўляецца вывучэнне і рэалізацыя алгарытмаў для сегментацыі будынкаў на здымках.

У выніку даследавання атрыманы наступныя **рэзультаты**:

- Рэалізаваны алгарытм з выкарыстаннем нейронная сеткі для выяўлення будынкаў на здымках.
- Праведзена ацэнка эфектыўнасці алгарытма і параўнанне з іншымі існуюычмі падыходамі рашэння дадзенай задачы
- Выяўлены шляхі далейшага ўдасканальвання алгарытма

Вобласцю ужывання з’яўляюцца задачы звязаныя з сегментацыяй здымкаў, алгарытм можна абагуліць на розныя дыскрэтныя аб’екты.

РЕФЕРАТ

Магистерская работа содержит:44 страницы, 22 иллюстрации, 1 приложение, 16 использованных источников.

Ключевые слова: глубокие нейронные сети, распознавание зданий, сегментация, градиентный спуск, функция стоимости, функция активации, архитектура сети.

Объектом исследования является использование нейронных сетей для выделения зданий на изображения земной поверхности.

Целью магистерской диссертации является изучение и реализация алгоритмов для сегментации зданий на изображениях

В результате исследования были получены следующие **результаты**:

- Реализован алгоритм с использованием нейронных сетей для выделения зданий на снимках
- Проведена оценка эффективности алгоритма и сравнение с другими существующими подходами решения данной задачи
- Изучены способы дальнейшего улучшения алгоритма

Областью применения являются задачи связанные с сегментацией изображений, алгоритм можно обобщить на различные дискретные объекты

ABSTRACT

The master's work contains 44 pages, 22 illustrations, 1 annex, 16 sources used.

Keywords: deep neural networks, building recognition, segmentation, gradient descent, cost function, activation function, network architecture.

The object of the study is the use of neural networks for the recognition of buildings on images of the earth's surface.

The aim of the master's thesis is to study and implement algorithms for segmenting buildings in images.

As a result of the research the following **results** were obtained:

- Implemented an algorithm using neural networks for recognition buildings on images.
- Evaluation of the efficiency of the algorithm and comparison with other existing approaches to solving this problem
- Methods for further improving the algorithm

The area of application is the problems associated with images segmentation, the algorithm can be generalized to various discrete objects

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ О НЕЙРОННЫХ СЕТЯХ

В данной главе рассмотрим теоретические сведенья о нейронных сетях, основные понятия нейронных сетей и алгоритмы на которые опирается дальнейшая работа.

1.1 Использование нейронных сетей

Наиболее сложные умственные задачи для человека являются простыми для компьютера. Компьютеры смогли давно обыграть лучших игроков в шахматы. Но только недавно приблизились к способностям людей распознавать объекты. Каждый день в жизни человека требуется большой объём знаний об окружающем мире. Большинство из этих знаний являются интуитивными и их тяжело представить формальным набором команд для компьютера. Компьютер должен как-то оперировать этими знаниями, чтобы вести себя разумно. Например, задача распознавания цифр является простейшей для человека. Но описать её формальным набором правил очень непросто.

Такие задачи решаются через обучение компьютера. Машинное обучение – раздел искусственного интеллекта, изучающий алгоритмы, способные обучаться. Очень распространёнными при решении задач машинного обучения являются нейронные сети.

Подход с использованием нейронных заключается в использовании большого набора тренировочных данных и разработке системы (модели), которая сможет обучиться на основе предоставленных тренировочных данных. То есть нейронная сеть сама построит правила для получения результата (классификация изображений, сегментаций изображений). И чем больше тренировочных данных используется для обучения нейронной сети, тем лучше будет результат. В тоже время нейронная сеть может обучиться хорошо работать с тренировочными данными, но давать совершенно неверный результат на данных из реальной жизни. Главная задача при обучении нейронной сети сделать так, чтобы обученная модель давала верный или близкий к верному результат на других данные, а не только на тех, которые использовались при тренировке. Эффект, когда нейронная сеть обучается слишком сильно на тренировочных данных и теряет качество на реальных тестовых данных, называется переобучением. Переобучение является одной из основных проблем машинного обучения.

Прежде чем заниматься обучением нейронной сети требуется собрать хороший массив данных. При этом качество результата непосредственно зависит от разнообразия и количества данных. Исходные данные как привило разделяются на несколько множеств. Тренировочное множество – данные, использующиеся непосредственно для обучения нейронной сети. Валидационное множество – данные, использующиеся для проверки работы

нейронной сети в процессе обучения. Важно понимать в процессе обучения насколько хорошо модель нейронной сети работает с данными на которых она не тренировалась. Такое распределение исходного массива помогает выявить переобучение на ранних этапах.

После того как выбраны данные для обучения следует переходить к построению архитектуры нейронной сети, заданию и изменению её параметров, экспериментированию. После обучения нейронной сети и получения некоторой модели переходят к определению качества модели на тестовых (реальных) данных.

Дальше рассмотрим базовые принципы работы нейронной сети.

1.2 Базовые понятия нейронных сетей

1.2.1 Перцептрон

Перцептрон – это математическая модель восприятия информации мозгом, разработанная Фрэнком Розенблаттом в 1957 году. Перцептрон стал одной из первых моделей нейросетей [1].

Перцептрон принимает на вход несколько бинарных значений и производит единственный бинарный выход.

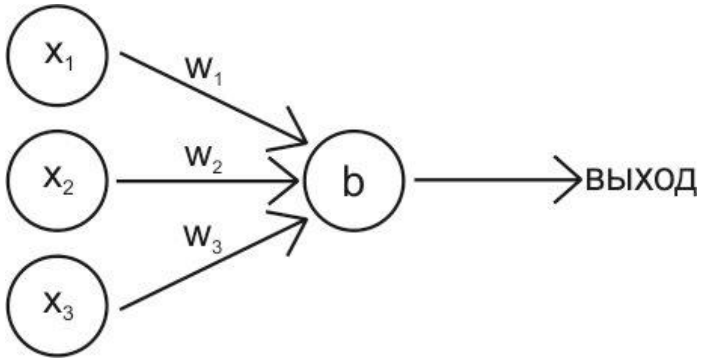

Рисунок 1.1

На Рисунок 1.1 перцептрон получает на вход три элемента x_1, x_2, x_3. Результат вычисляется по формуле (1.1):

$$g(z) = g\left(\sum_{i=1}^{3} w_i x_i + b\right) = \begin{cases} 0, \sum_{i=1}^{3} w_i x_i + b \leq 0 \\ 1, \sum_{i=1}^{3} w_i x_i + b > 0 \end{cases}, \quad (1.1)$$

Где w_1, w_2, w_3 – веса, действительные числа, выражающие важность каждого элемента входа, его влияние на выход, b – смещение, пороговое значением в зависимости от которого изменяется выход.

Перцептрон представляет собой ступенчатую функцию Рисунок 1.2.

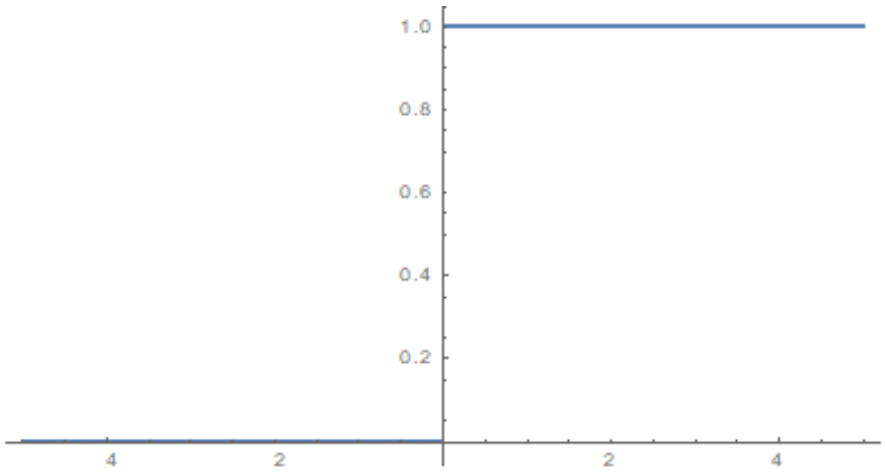

Рисунок 1.2

Выше показан только один нейрон. Саму же нейронную сеть можно представить объединением множества таких нейронов Рисунок 1.3, где выход одного нейрона является входом следующего, и нейрон на каждом последующем уровне зависит от нейронов на предыдущих.

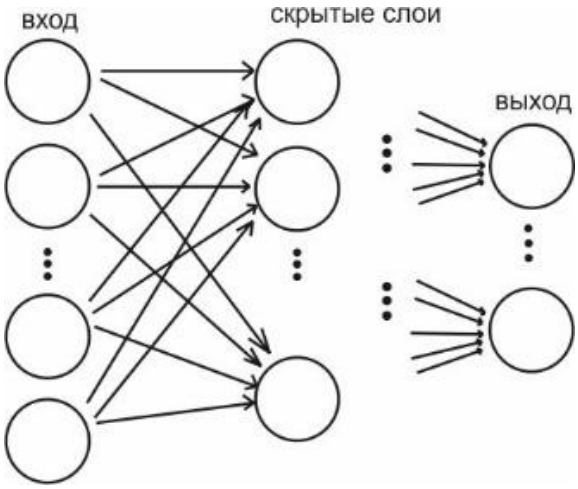

Рисунок 1.3

Скрытые слои представляют структуру обучающего алгоритма. Цель нейронной сети – приблизить некоторою функцию *f*. Подобрать в процессе обучения такие веса и смещения, чтобы максимально точно приближать необходимый результат.

Введём обозначения, которые будут использоваться далее. В данной работе будем рассматривать нейронные сети где выход одного слоя является входом следующего. Т.е. в сети нету циклов и информация распространяется только в одном направлении. Такие нейронные сети называются *feedforward*. Также каждый нейрон некоторого слоя связан со всеми нейронами из предыдущего слоя. Количество слоёв нейронной сети: *L* (первый слой входа не учитываем).

Вход – вектор столбец $x = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix}$, где n – количество элементов входа. Веса соединяющие слой l и $l+1$ задаются матрицей $w^l = \begin{pmatrix} w_{11}^l & ... & w_{1n}^l \\ ... & ... & ... \\ w_{m1}^l & ... & w_{mn}^l \end{pmatrix}$, где n – количество нейронов слоя $l+1$, m – количество нейронов слоя l. Т.е. w_{ij}^l – вес между нейронами i и j слоёв $l+1$ и l соответственно. Смещение слоя l задаёт вектор столбец $b^l = \begin{pmatrix} b_1^l \\ b_2^l \\ ... \\ b_n^l \end{pmatrix}$, где n – количество нейронов в слое l. Введём обозначение $z^1 = w^1 * x + b^1$, чтобы получить значения первого скрытого слоя необходимо воспользоваться (1.1) или другой функцией (далее будут описаны разные функции активации). Значения нейронов первого скрытого слоя обозначим через $a^1 = g(z^1)$. Тогда вычислять значения каждого последующего слоя можно по формуле $a^l = g(z^l) = (w^l * a^{l-1} + b^l)$. Итоговый выход из нейронной сети будет: $a^L = g(z^L)$.

Общий процесс вычисления значений нейронной сети можно представить (1.2):

$$\begin{aligned} a^1 &= g(w^1 x + b^1), \\ a^2 &= g(w^2 a^1 + b^2) \\ &\dots \\ a^L &= g(w^L a^{L-1} + b^L) \end{aligned} \tag{1.2}$$

Где g – некоторая функция, преобразующая один слой в другой. Это может быть, как перцептрон, так и другая функция. Перцептрон – самая простая модель нейрона, к его недостаткам относятся разрывность и бинарность. Так небольшое изменение весов может сказаться на изменении выхода в противоположную сторону. Перцептрон был приведён как историческая справка первого искусственного нейрона. Далее в данной работе будут использоваться более сложные функции активации.

1.2.2 Функция стоимости

Функция стоимости используется при обучении нейронной сети. И помогает понять насколько хорошо обучена нейронная сеть. Изменяя на каждом шаге обучения нейронной сети веса и смещения можно наблюдать как изменяется функция стоимости. Задача обучения заключается в минимизации функции стоимости.

В разных ситуациях применяются разные функции стоимости. Рассмотрим некоторые из них.

Предварительно введём обозначения. $C(W,b)$ – функция стоимости, W – веса, используемые в нейронной сети, b – смещения нейронной сети, x – вход нейронной сети, X – все входные данные для сети, a– выход нейронной сети, $y(x)$ – истинный результат для входных данных x. Необходимо подобрать такие значения W и b, что $C(W,b) \to min$.

Одна из наиболее простых функций стоимости (1.3) *mean squared error (MSE)* или средняя квадратичная ошибка.

$$C(W,b) = \frac{1}{2n}\sum_{x\in X}\|y(x)-a\|_2^2 \tag{1.3}$$

Где $\|x\|_2 = \sqrt{\sum_i x_i^2}$ – длина вектора. $C(W,b) \geq 0$.

Двойку в знаменателе удобно использовать для вычисления (при вычислении производной она сокращается).

Если результат обучения a близок к значению $y(x)$, то функция стоимости $C(W,b) \approx 0$, т.е в этом случае алгоритм сделал хорошую работу и нейронная сеть хорошо приближает требуемую функцию, напротив если $C(W,b)$ будет большим, то значит $y(x)$ и a сильно отличаются и нужно продолжать обучать нейронную сеть.

Другой пример функции стоимости (1.4) – *cross entropy*.

$$C(W,b) = -\frac{1}{n}\sum_x\sum_j (\, y_j \ln a_j + (1-y_j)\ln(1-a_j)) \tag{1.4}$$

Где y_j, a_j – элементы вектора истинного значения и выхода нейронной сети соответственно для входных данных x. Можно заметить, что $C(W,b) > 0$, также если истинный результат $y_j = 1$ или $y_j = 0$ (так обычно в задачах классификации) и значение выхода нейронной сети близко к необходимому значению следует $C(W,b) \to 0$. Ещё, функция *cross entropy*, имеет то преимущество, что, в отличие от *MSE*, она позволяет избежать проблем замедления обучения [1].

В целом использование определённой функции стоимости зависит от задачи, функций активации и архитектуры сети. Функция *cross entropy* предпочтительна в задачах классификации, а *MSE* в задачах регрессии.

1.2.3 Функция активации

Значения нейронов скрытых слоёв вычисляются с помощью некоторой функции. Выбор такой функции сильно влияет на результат сети. Эта функция называется функцией активации, можно сказать, что она активирует (задаёт значение) последующим нейронам. Обычно невозможно предсказать заранее

какая функция подойдёт больше. Процесс проектирования нейронной сети, выбора функций состоит из проб и ошибок. Тренировка сети и проверка на валидационном множестве позволяют выявить результат и выбрать лучший. С самой простой такой функцией (перцептрон) мы познакомились выше (1.1). Сейчас приведём несколько других примеров функции активации.

Одна из наиболее часто используемых функций – *сигмоид*.

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.5}$$

Сигмоид – непрерывная, дифференцируемая, при этом ограничена горизонтальными асимптотами $y = 0, y = 1$

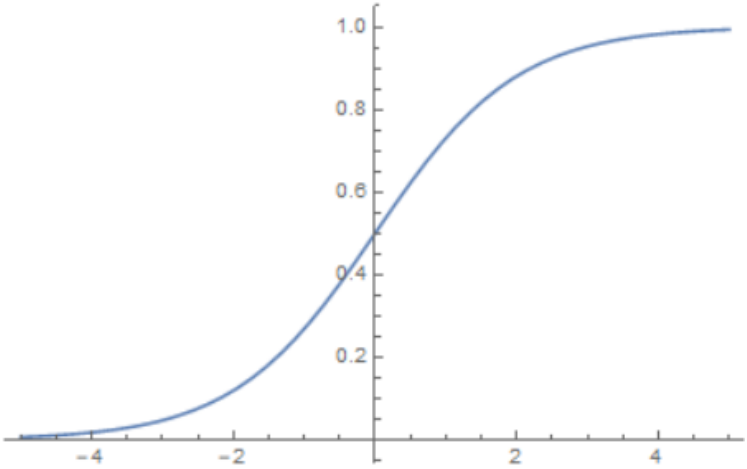

Рисунок 1.4

Подходит в случаях, когда необходимо задавать вероятность некоторого события.

Другая функция, похожая на сигмоид – *гиперболический тангенс*.

$$g(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{1.6}$$

$tanh(z) = 2\sigma(2z) - 1$, является сдвинутой и масштабированной версией сигмоида.

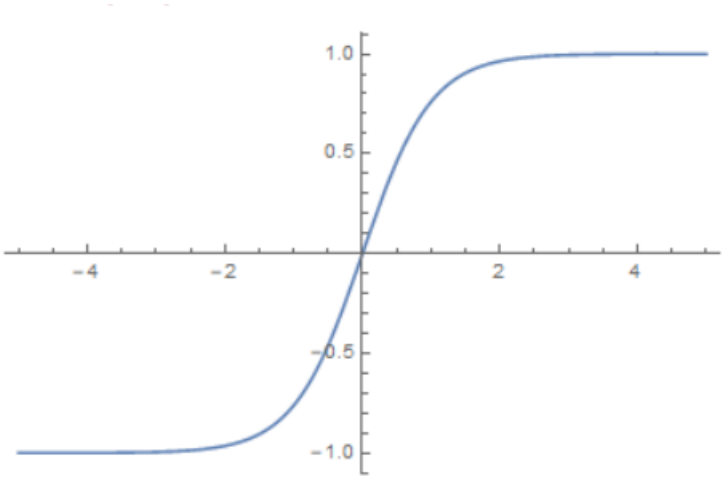

Рисунок 1.5

Rectified linear units (Relu):

$$g(z) = \max(0, z) \tag{1.7}$$

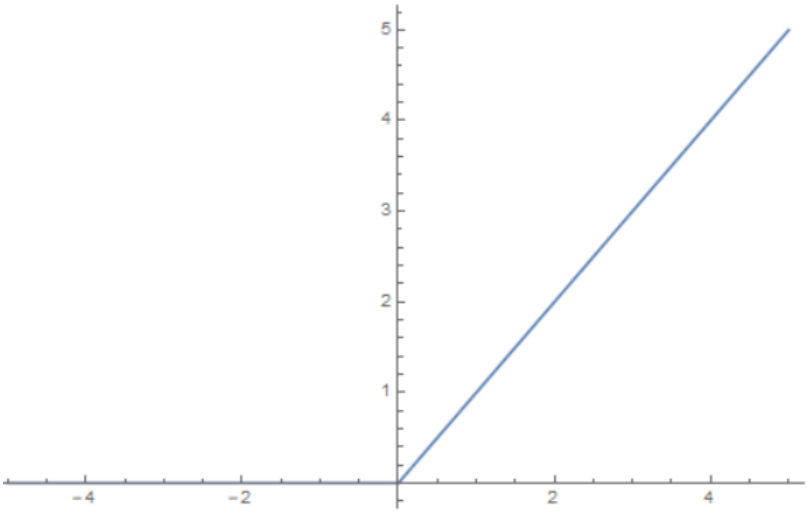

Рисунок 1.6

Отличие от линейной функции в том, что она возвращает 0 на половине своей области определения. Relu является простой в оптимизации. Функции сигмоида и гиперболического тангенса имеют свойство насыщения, т.е., когда, значение сигмодида близко к 0 или 1, процесс обучения замедляется. В тоже время для relu нету такой проблемы – увеличение значений входа relu не вызывает соответствующего замедления в обучении. Также relu более просто вычисляется математически чем сигмоид или гиперболический тангенс.

Exponential linear units (Elu):

$$g(z) = \begin{cases} e^z - 1, z < 0 \\ z, z \geq 0 \end{cases} \quad (1.8)$$

Elu ускоряет обучение глубоких нейронных сетей и приводит к более высокой точности классификации. Как и relu данная функция смягчает проблемы насыщения градиента через тождественную функцию для положительных значений. В отличие от relu, Elu имеют отрицательные значения, что помогает держать среднее значение активации ближе к 0. Кроме того, сети, использующие elu, дали конкурентные результаты на данных ImageNet за гораздо меньшее количество эпох, чем соответствующая сеть с relu [5].

1.3 Базовые алгоритмы нейронных сетей

1.3.1 Градиентный спуск

Задача обучения нейронной сети – минимизация функции стоимости, т.е. минимизация функции многих переменных. Так как функция, которую необходимо минимизировать достаточно сложная и обычно имеет очень большое число параметров, обычный способ по поиску экстремума функции не подойдёт. Сперва нужно подсчитать производные первого порядка, решить систему уравнений и найти точки подозрительные на экстремум. Затем для того чтобы проверить какая из точек является локальным минимум нужно вычислить производные второго порядка. Такой способ очень затратный по вычислению и не подойдёт. Поэтому алгоритмы минимизации функции, применяемые в машинном обучении, в основном базируются на градиентном спуске.

Рассмотрим алгоритм градиентного спуска. Для простоты вычислений выберем функцию стоимости (1.3). Наша задача состоит в минимизации функции многих переменных, функции весов и смещений.

Определение 1 Глобальный минимум функции $f: X \to \mathbb{R}$, $x_0 \in \min f <=>$ $\forall x \in X$ выполняется $f(x) \geq f(x_0)$

Определение 2 Локальный минимум функции $f: X \to \mathbb{R}$, $x_0 \in loc \min f <=>$ $\exists O_\epsilon(x_0)$, что $\forall x \in O_\epsilon(x_0)$ выполянется $f(x) \geq f(x_0)$

Определение 3 Градиентом функции $f: \mathbb{R}^n \to \mathbb{R}$ называется вектор частных производных $\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n})$.

Определение 4 Производная функции $f: \mathbb{R}^n \to \mathbb{R}$ по направлению e называется $\frac{\partial f}{\partial e}(x) = \lim_{h \to 0} \frac{f(x+e*h)-f(x)}{h}$. Является проекцией градиента на это направление.

Пусть наша задача состоит в минимизации функции $f: \mathbb{R}^n \to \mathbb{R}$.

Для минимизации функции f мы хотим найти направление в котором f уменьшается быстрее всего. Т.е. необходимо найти $\min e \nabla f =$

$\min\|e\|\|\nabla f\|\cos\theta$, считая длину вектора постоянной, выражение сведётся к $\min\cos\theta$, $\theta = \pi$ т.е. значение производной по направлению минимально если её направление противоположно направлению вектора градиента.

Таким образом мы можем уменьшать нашу функцию в направлении обратную градиенту. По следующему правилу:

$$x_i' = x_i - \eta\frac{\partial f}{\partial x_i} \tag{1.9}$$

Где η- learning rate, скорость обучения, положительное число, задающее размер шага, с которым заданная функция уменьшается. Обычно в качестве η выбирается небольшое положительное число. Так мы можем уменьшать значение функции, применяя это правило несколько раз подряд.

Перенесём (1.9*)* на функцию стоимости, получим:

$$w_i' = w_i - \eta\frac{\partial C}{\partial w_i} \tag{1.10}$$
$$b_i' = b_i - \eta\frac{\partial C}{\partial b_i}$$

Так как функция стоимости (1.3) – усреднённая по всем входам, то для вычисления новых значений w_i' и b_i' нам придётся вычислять градиент ∇C для каждого входа, а затем усреднять его. Иногда число таких входов может быть большим. И такое вычисление займёт очень много времени. Поэтому, удобнее производить вычисления градиента на случайных небольших наборах данных (нам не нужно знать точное уменьшение функции, нам главное двигаться в нужном направлении). Такие входы называют *mini batch.* При этом мы полагаем, что значение градиента на таких партиях примерно такое же, как и на всех данных. Такой алгоритм обучения называется *стохастическим градиентным спуском*.

Например, всё множество входных данных X случайно разбивается на несколько подмножеств: $X_1, X_2, \ldots X_k$, пусть при этом размер каждого подмножества m. Далее алгоритм тренируется на каждом из этих подмножеств, т.е.

$$w_i' = w_i - \eta\frac{\partial C_{X_j}}{\partial w_i} \tag{1.11}$$
$$b_i' = b_i - \eta\frac{\partial C_{X_j}}{\partial b_i}$$

Где C_{X_j}- функция усреднённая по набору X_j. Значнеия весов и смещений будут изменяться после каждой тренировки на наборах X_j. Набор тренировок по всем партиям данных X_j называется эпохой. После того как одна эпоха закончилась данные снова разделяются на подмножества и происходит обучение на каждом из подмножеств, т.е. начинается следующая эпоха.

Обратим внимание на возможные проблемы градиентного спуска. На Рисунок 1.7 представлена функция одной переменной с несколькими минимумами.

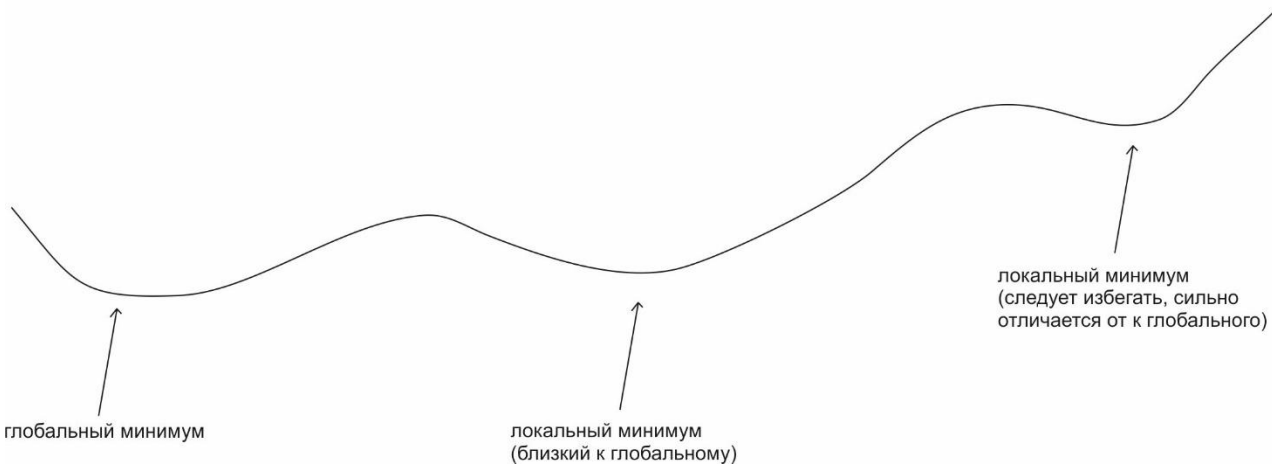

Рисунок 1.7

Если при работе алгоритма градиентного спуска начальное положение значения весов и смещения были выбраны рядом с одним из локальных минимумов, то существует вероятность того, что мы не сможем выбраться из этого локального минимума. И если значение этого локального минимума сильно отличается от глобального, то и результат работы нейронной сети будет далёк от желаемого.

При глубоком обучении мы минимизируем функцию многих переменных, которая имеет множество локальных минимумов, седловых точек, окружённых плоской областью. Это делает оптимизацию достаточно сложной. Поэтому мы обычно соглашаемся на поиск значения функции, которое является очень низким, но не обязательно является глобальным минимумом.

1.3.2 Алгоритм обратного распространения ошибки

В то время, когда алгоритм градиентного спуска служит для минимизации функции, алгоритм обратного распространения ошибки предназначен для быстрого вычисления градиента. Для вычисления градиента необходимо вычислять частные производные: $\frac{\partial C}{\partial w_{ij}}$, $\frac{\partial C}{\partial b_i}$. Вычислять их каждый раз вручную трудная задача, так как нейронная сеть может быть глубокая и иметь множество параметров. При этом необходимо это делать быстро. Для быстрого вычисления градиента функции стоимости был придуман алгоритм обратного распространения ошибки. Метод был впервые придуман в 1970-е годы, далее существенно развит в 1986. Описание присутствует в [1], [2]. Рассмотрим его далее.

Символом $\odot$ будем обозначать операцию Адамара.

Определение 5 - Операция Адамара – операция над двумя матрицами одинаковой размерности, результатом которой является матрица такой же размерности. Значение элемента с индексом *ij* получается, как произведение элементов с таким же индексом исходных матриц.

Воспользуемся обозначениями (1.2) данными выше. В качестве функции стоимости используем среднюю квадратичную ошибку (1.3), в качестве функции активации сигмоид (1.5). Введём ошибку нейрона *j* слоя *l*:

$$\delta_j^l = \frac{\delta C}{\delta z_j^l} \tag{1.12}$$

Тогда общий алгоритм будет следующим.

Алгоритм обратного распространения ошибки:

1. Считаем активацию для каждого из слоёв нейронной сети

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{1.13}$$

2. Вычисляем вектор ошибки последнего слоя

$$\delta^L = \nabla C \odot \sigma'(w^L a^{L-1} + b^L) \tag{1.14}$$

3. Распространяем ошибку назад, для каждого слоя *l=L, L-1,..2*

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(w^{l-1} a^{l-1} + b^l) \tag{1.15}$$

4. Вычисляем градиент функции стоимости по формулам:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \frac{\partial C}{\partial w_{ij}^l} = a_j^l \delta_j^l, \tag{1.16}$$

Алгоритм называется обратным распространением ошибки, так как ошибка распространяется назад, на первые слои сети. Мы вычисляем ошибку начиная с последнего слоя и заканчивая первым. И вычисляем все необходимые параметры нейронной сети за два прохода.

Разберём некоторые пункты алгоритма более подробно.

$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\,\sigma'(z_j^L)$, переписывая в векторном виде получим (1.14).

Покажем, как из ошибки *l+1* слоя можно получить ошибку слоя *l.*

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{1.17}$$

$z_k^{l+1} = w_k^{l+1} a^l + b^k = \sum_i w_{ik}^{l+1}\sigma(z_i^l) + b^k$, продифференцируем это выражение по z_i^l получим: $\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l)$, тогда, подставляем это выражение в (1.17) получим $\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1}\sigma'(z_j^l) = \sigma'(z_j^l)\,(w_j^{l+1})^T\delta^{l+1}$, переписав в векторном виде получим (1.15).

Осталось рассмотреть (1.16). Распишем $\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l}$, т.к. $z_j^l = \sum_i w_{ij}^l * a_i^{l-1} + b_j^l$, то $\frac{\partial z_j^l}{\partial b_j^l} = 1$, получаем $\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \delta_j^l$

Аналогично рассматривается и второе уравнение ,$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ij}^l}$, где $z_j^l = \sum_k w_{kj}^l * a_k^{l-1} + b_j^l$, тогда $\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^{l-1}$ и общий результат $\frac{\partial C}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}$.

Выше приведён алгоритм в случае одного входа в сеть. Когда мы используем стохастический градиентный спуск таких входов несколько. В этом случае мы вычисляем ошибку (1.12) для каждого входа сети. И в конце изменяем веса и смещения по следующему правилу:

$$w_{ij}' = w_{ij} - \frac{\eta}{m}\sum_x a_j^{x,l}\delta_j^{x,l}, \tag{1.18}$$

$$b_j' = b_j - \frac{\eta}{m}\sum_x \delta_j^{x,l},$$

Где m – количество элементов в мини батче.

1.4 Некоторые замечания

Теперь, когда мы знаем базовые обучающие алгоритмы нейронных сетей. Сделаем несколько выводов из полученных знаний.

Рассмотрим простой пример для одного нейрона Рисунок 1.1, в качестве функции стоимости используется средняя квадратичная ошибка (1.3), а в качестве функции активации – сигмоид. Будем вычислять значение градиента. Получим:

$$\frac{\partial C}{\partial w_{ij}} = \frac{1}{2n}\sum_x (\sigma(z) - y)\sigma'(z)x_j \tag{1.19}$$

Мы видим, что значения элементов вектора градиента зависят от производной сигмоида. А $\sigma'(z)$ принимает небольшие значения, когда $\sigma(z)$ близка к 0 или 1. Это может значительно замедлить обучение. Если вместо $\sigma(z)$ мы будем использовать relu, такого не произойдёт, так как производная для relu постоянна и равна единице.

Теперь заменим функцию средней квадратичной ошибки на *cross entropy* (1.4). В этом случае получим:

$$\frac{\partial C}{\partial w_{ij}} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right)\sigma'(z)x_j \tag{1.20}$$

$$= -\frac{1}{n}\sum_x \frac{y-\sigma(z)}{\sigma(z)(1-\sigma(z))}\sigma'(z)x_j$$

Распишем $\sigma'(z) = \left(\frac{1}{1+e^{-z}}\right)' = \frac{e^{-z}}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right) * \left(\frac{e^{-z}+1-1}{1+e^{-z}}\right) = \sigma(z) * \left(\frac{-1}{1+e^{-z}} + \frac{1+e^{-z}}{1+e^{-z}}\right) = \sigma(z)(1-\sigma(z))$ и подставим в .(1.20) получим:

$$\frac{\partial C}{\partial w_{ij}} = \frac{1}{n}\sum_x (\sigma(z) - y)x_j \tag{1.21}$$

Получилось, что значение вектора градиента не зависит от производной сигмоида, поэтому замедление обучения, связанное с малым значением производной, в данном случае исчезает. Это наиболее важное свойство почему функция cross entropy и была выбрана. Можно заметить ещё одно свойство, что скорость обучения зависит от $\sigma(z) - y$ т.е. ошибки в выходе из сети. Чем больше ошибка, тем быстрее нейрон будет обучаться.

Здесь мы привели пример только для весов аналогичные рассуждения можно провести и для смещения.

1.5 Свёрточные нейронные сети

В предыдущих разделах были рассмотрены нейронные сети прямого распространения. Это самый простой случай. Тем не менее алгоритмы градиентного спуска и обратного распространения ошибки используются для любых нейронных сетей. Хотя сами вычисления усложняются с выбором более сложных нейронных сетей.

Далее рассмотрим принцип работы свёрточных нейронных сетей. Благодаря своей структуре свёрточные нейронные сети очень часто применяются для классификации и сегментации изображений, при этом показывают хорошие результаты.

Свёрточные сети получили своё название, т.к. используют внутри операцию свёртки.

Далее будем рассматривать свёрточные нейронные сети применительно к изображениям. Один слой свёрточной нейронной сети лучше воспринимать как квадрат нейронов размером *n*n.*

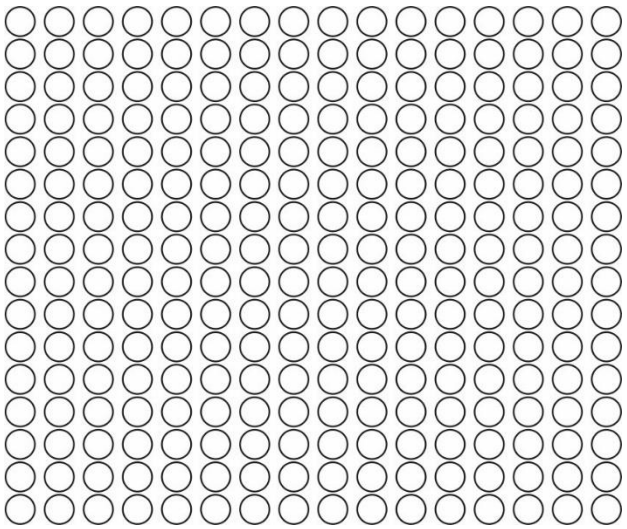

Рисунок 1.8

На Рисунок 1.8 представлено изображение размером 16x16. Где каждому пикселю изображения соответствует один нейрон.

1.5.1 Операция свёртки

Определение 6 Свёртка – особый вид интегрального преобразования. Пусть f и g – две интегрируемые функции $R \to R$, тогда свёрткой называется:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t)g(x-t)dt \tag{1.22}$$

Операция свёртки обладает свойствами:

1. *коммутативность*: $(f * g) = (g * f)$
2. *ассоциативность*: $(f * (g * h)) = ((f * g) * h)$
3. *дистрибутивность:* $(f * (g + h)) = (f * g) + (f * h)$

Рассмотрим операцию свёртки на примере. Пусть стоит задача вычислить количество нерастаявшего снега в определённый момент времени. Сложность заключается в том, что во время выпадения нового снега, старый может таять. При этом нам известно *f(t)* – количество снега которое выпадает в конкретный момент времени. И *g(Δt)* – доля нерастаявшего снега через промежуток времени *Δt*. (Доля от количества снега *f(t)*). Введём другие обозначения: *x* – момент времени в который мы хотим узнать количество оставшегося снега, t – момент выпадения конкретной порции снега. Необходимо построить функцию *s(x)*.

Тогда, например, если нам необходимо найти количество снега в момент времени x, оставшееся от снега выпавшего в момент времени t, то *s(x) = f(t)*g(x-t)*. И если мы просуммируем по всему выпавшему ранее снегу, мы получим в точности формулу (1.22).

Очень часто данные представляются через определённый временной интервал, например, одна секунда. В этом случае мы работаем с дискретной свёрткой:

$$(f * g)(x) = \sum_{-\infty}^{+\infty} f(t)g(x-t) \tag{1.23}$$

Теперь рассмотрим основные принципы, на которых основаны свёрточные нейронные сети. Затем вернёмся к применению операции свёртки для изображений.

1.5.2 Локальные области (local receptive field)

Как и в сетях прямого распространения мы соединяем нейроны входа с нейронами скрытого слоя. Однако вместо того, чтобы соединять каждый нейрон входа с каждым нейроном скрытого слоя. Будем соединять только нейроны из конкретной маленькой области входа с одним нейроном скрытого слоя. При этом каждая связь имеет вес и нейрон скрытого слоя имеет своё смещение.

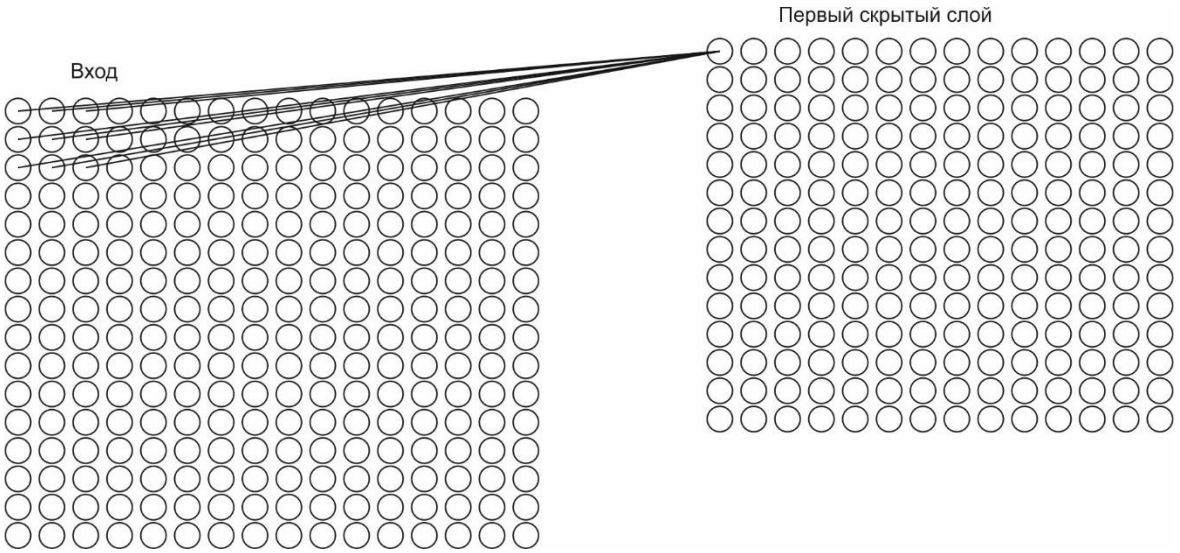

Рисунок 1.9

На Рисунок 1.9 показаны связи нейронов локальной области входного изображения с одним нейроном скрытого слоя. Здесь размер локальной области равен 3x3. Для остальных связей локальная область, смещается, например, на один нейрон вправо или вниз и соединяется с другим нейроном скрытого слоя. Так пока все нейроны не будут соединены. Так же определяются связи других свёрточных слоёв нейронной сети.

1.5.3 Общие параметры

Для одного нейрона скрытого слоя используются те же самые веса и смещение, что и для остальных. Это означает, что все нейроны в первом скрытом слое будут выделять тоже самое свойство, но только в разных местах изображения. Если предположить, что нейрон скрытого слоя выделяет вертикальное ребро в конкретной локальной области, то и в других областях

изображения он будет выделять такое же. Это особенность помогает если выделяемый объект сдвигается на изображении.

В терминологии свёрточной сети мы называем результат операции свёртки, т.е. результат отображения входного слоя в скрытый – *feature map*. Веса и смещения являются общими. И вместе называются *kernel* или *filter*. Для того, чтобы распознать изображение хорошо, нам необходима больше чем одна *feature map*. Поэтому целиком один слой свёрточной нейронной сети состоит из нескольких *feature map*.

Определим активацию нейрона исходя из данных свойств и Рисунок 1.9.

$$a_{i,j}^{1} = g(\sum_{n=1}^{3}\sum_{m=1}^{3} w_{n,m}^{1} * x_{i+n-1,j+m-1} + b) \tag{1.24}$$

Где *g(z)* – функция активации, $w_{n,m}^{1}$, *n* и *m* от 1 до 3 – общие веса, *b* – общее смещение.

Воспользуемся (1.23) где определим функции f и g равными 0 на всей области определения кроме конечного множества точек. И поскольку мы работаем с изображениями, то есть вход имеет размерность два. Мы можем переписать (1.23) к виду:

$$(I * K)(i,j) = \sum_{n=1}^{N}\sum_{m=1}^{M} I(n,m) * K(i-n, j-m) \tag{1.25}$$

Где *I* – изображение, *K* – kernel, так как операция свёртки – коммутативная, то верно:

$$(K * I)(i,j) = \sum_{n=1}^{N}\sum_{m=1}^{M} I(i-n, j-m) * K(n,m) \tag{1.26}$$

Сразу введём обозначения для любого свёрточного слоя: $K(n,m) = w_{n,m}^{l}$ и $I(i-n, j-m) = a_{i+n-1,j+m-1}^{l-1}$

Тогда (1.24) для в общем виде преобразуется к (1.27):

$$a^{l} = g((a^{l-1} * w^{l})(i,j) + b^{l}) \tag{1.27}$$

Тут определена активация для выделения одного свойства. При определении нескольких *feature map* будет выделено несколько групп общих весов и смещений. И полностью представить скрытый слой свёрточной нейронной сети можно так:

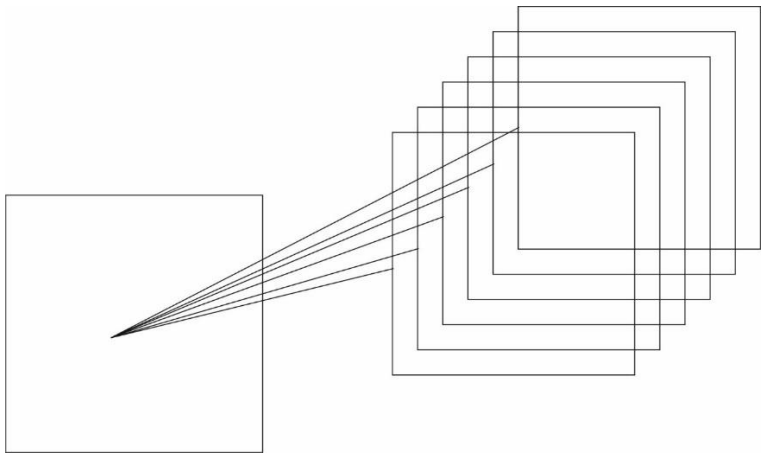

Рисунок 1.10

На данном рисунке для одного входного слоя нейронной сети генерируется 6 *feature map*. Хотя на практике используются значения больше.

Также преимуществом использования общих весов и смещения является сокращение большого числа параметров чем в нейронных сетях где каждый нейрон соединён с каждым из последующего слоя. Например, для Рисунок 1.9 у нас будет 1960 параметров, против 65792 для обычной сети. Получаем выигрыш почти в 30 раз, что даёт выигрыш в памяти и скорости обучения.

1.5.4 Слои объединения (Pooling layers)

Функция объединения заменяет выходные данные сети в определенном месте суммарной статистикой соседних выходов. Например, максимальное объединение (*max pooling*) сообщает о максимальном выходе в прямоугольной окрестности. Другие популярные функции объединения включают среднее значение прямоугольной окрестности. Например, норма L2 прямоугольной окрестности или средневзвешенное значение, основанное на расстоянии от центрального пикселя. Во всех случаях объединение помогает сделать результат инвариантным к небольшим смещениям входа. Инвариантность обозначает, что, если мы сместим вход на небольшое значение объединённый выход не изменится. Это является важным в случае, когда нам необходимо знать присутствует ли конкретное свойство, чем конкретное место его присутствия. Например, если мы хотим определить содержит ли изображение лицо, нам не нужно знать положение глаз с точностью до пикселя. Нам достаточно убедиться, что присутствует глаз с левой стороны изображения и присутствует ли глаз с правой стороны изображения. В другом случае если нам необходимо найти точку пересечения двух рёбер, мы должны сохранить положение рёбер достаточно точно, чтобы убедиться встречаются они или нет. Исходя из этого свойства слоёв объединения и необходимо их применять. Добавим к результату сети на Рисунок 1.9 один слой объединения:

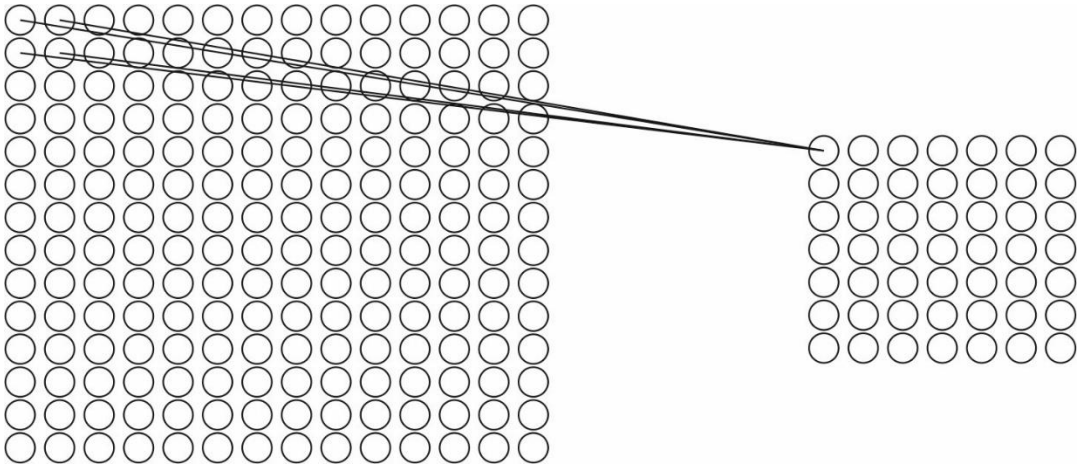

Рисунок 1.11

На Рисунок 1.11 после свёрточного слоя нейронной сети применяется слой объединения к области размером 2x2. После такого объединение размер изображения уменьшается в два раза.

Также как упоминалось ранее один слой свёрточной нейронной сети содержит несколько *feature map*, то функция объединения будет применяться к каждой из них отдельно.

1.5.5 Общая структура свёрточной нейронной сети

Хотя архитектура свёрточных нейронных сетей отличается от сетей прямого распространения, принцип работы остаётся таким же, используются те же алгоритмы градиентного спуска и обратного распространения ошибки. Нейронная сеть обучается на тренировочном множестве, в процессе обучения изменяются веса и смещения и цель: хорошо предсказывать результат. На Рисунок 1.12 представлен стандартный базовый блок свёрточной нейронной сети. Последовательность действий следующая: свёртка затем активации, затем объединение.

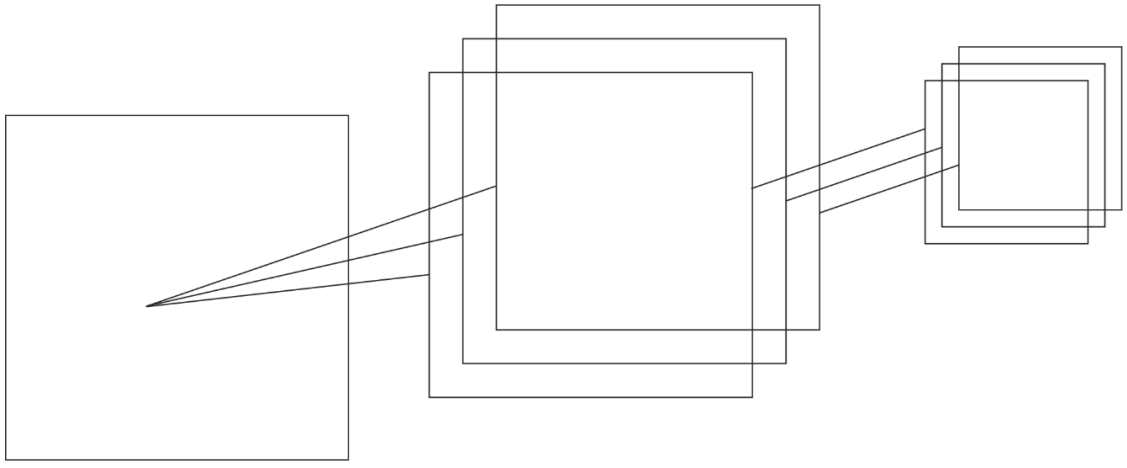

Рисунок 1.12

2 ВЫДЕЛЕНИЕ ЗДАНИЙ НА ИЗОБРАЖЕНИЯХ ЗЕМНОЙ ПОВЕРХНОСТИ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

В данной главе будет рассмотрен подход по сегментации зданий на спутниковых изображениях. Разберём существующие решения для распознавания зданий, рассмотрим архитектуру нейронной сети, используемую в данной работе и процесс обучения.

2.1 Анализ существующих алгоритмов сегментации зданий на изображениях

Прежде чем перейти к описанию разработанного алгоритма рассмотрим какие подходы и результаты в распознавании изображений, а именно сегментации зданий существуют на данный момент.

Ранее для выделения зданий на спутниковых изображениях использовались алгоритмы, опирающиеся на анализ цветового контура зданий, анализ теней, материалов крыш зданий, форм зданий, их границ [7]. Сейчас такой подход уступил место свёрточным нейронным сетям и может использоваться больше как дополнительная проверка результата нейронных сетей.

Как упоминалось выше первые нейронные сети появились в середине 20-го века. Эффективный алгоритм обратного распространения ошибки был разработан в 80-х годах 20-го века. Но широкое использование нейронные сети получили только в последние 10 лет. Это связано с ростом вычислительных мощностей и возможностью осуществлять эксперименты в домашних условиях.

Работа [6] показывает эффективность использования глубоких нейронных сетей для классификации изображений и опережает все предшествующие результаты. Тренировка осуществлялась на больших объёмах данных, больше миллиона изображений. Нужно было классифицировать изображения на 1000 классов. Работа показывает, что, используя глубокую свёрточную нейронную сеть, можно достичь хороших результатов. Причём, выбрасывая, например, один слой свёртки производительность падает. Таким образом глубина сети является важной характеристикой.

Работа [8] демонстрирует подход, который был применён для соревнования SpaceNet по выделению зданий, проводимым TopCoder. На соревновании был предоставлен маркированный набор данных. Подход данной работы заключался в изменении маски изображения. Каждому пикселю присваивалось значение насколько далеко он отстоит от границы. При этом, если пиксель принадлежал внутренней части здания, значение было положительное, иначе отрицательное. Данные расстояния нормализировались от -1 до 1. После этого обучалась свёрточная нейронная сеть. Сеть состояла из нескольких слоёв

свёртки, после каждой свёртки в качестве функции активации использовалась relu (1.7), затем следовала батч нормализация [4]43. Было несколько слоёв максимального объединения 1.5.4. Итоговая часть алгоритма – это превращение матрицы расстояний в полигоны, определяющие границу здания.

Работа [9] посвящена сегментации изображений. И является результатом участия в соревновании DSTL Satellite Imagery Feature Detection, которое проводилось на kaggle. Целью было сегментировать 10 различных классов изображений. Подход заключался в использовании модифицированной свёрточной сети U-Net [10], аугументации исходных данных. Так как в данном соревновании изображения содержали 16 каналов, в данной работе для сегментации некоторых классов использовались свойства этих каналов. Т.е. классы воды и растительности можно было получить без предварительного обучения, только извлекая информацию из пикселей изображения, используя вегетационные индексы (NDWI, CCCI). Также было обращено внимание на проблему классификации граничных пикселей.

2.2 Предлагаемый алгоритм выделения зданий на изображениях земной поверхности

2.2.1 Нейронная сеть

Для сегментации и выделения зданий использовалась нейронная сеть U-Net [10]. Впервые U-Net была применена для сегментации нейронных структур нервной ткани личинки Дрозофилы. Использование U-Net для сегментации превзошло другие способы сегментации изображений. В частности, способ использующий скользящее окно [3], который предсказывал класс каждого пикселя предоставляя регион вокруг него как вход. У данного подхода существовало несколько недостатков таких избыточность информации из-за необходимости создания таких регионов для каждого пикселя (т.е. изображений для обучения становится гораздо больше чем исходных) и вопросы, связанные с размером данного региона. Область маленького размера предоставляет мало контекста для классификации, для большой области необходимо применять несколько max pooling слоёв, но так происходит потеря положения пикселя и уменьшается точность классификации.

После успеха с сегментацией нейронных структур, U-Net стала активно применяться для сегментации объектов на разных изображениях. В связи с этим в данной работе для сегментации зданий также используется U-Net.

Архитектура U-Net представлена на Рисунок 2.1:

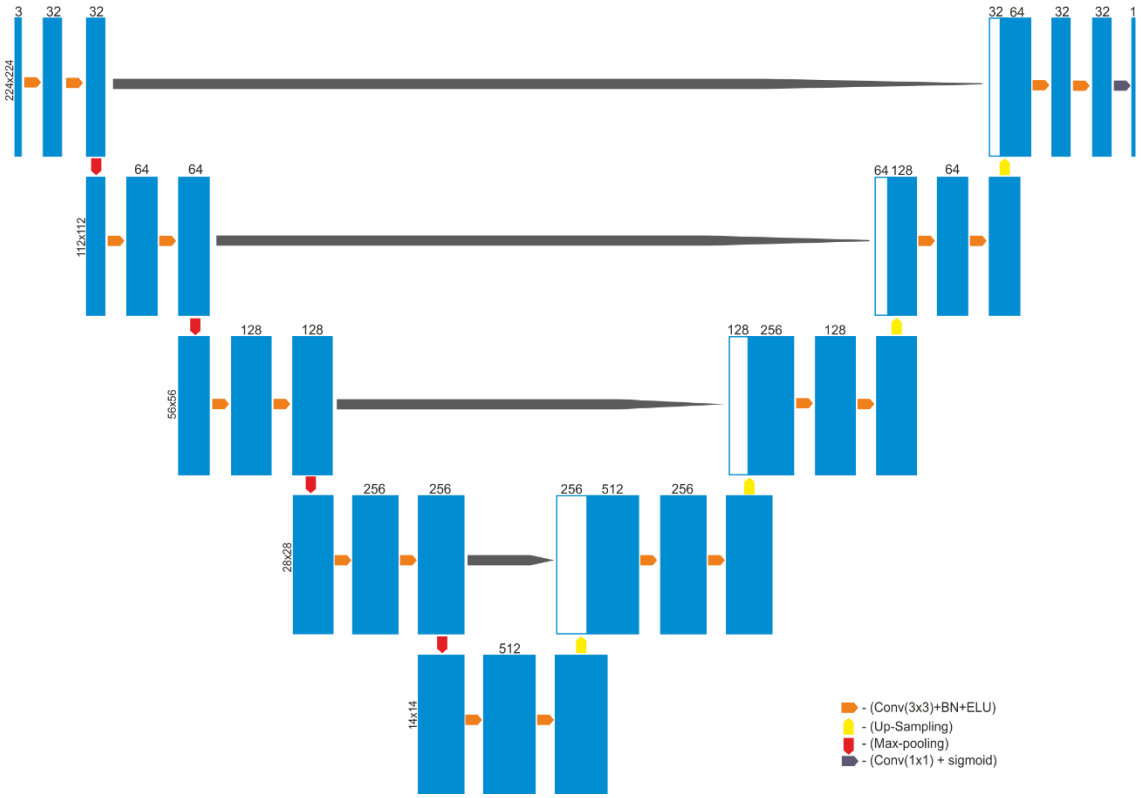

Рисунок 2.1

2.2.2 Стадии алгоритма

Рассмотрим архитектуру сети более детально. U-Net представляет собой свёрточную нейронную сеть. И состоит из двух частей: сжимающаяся и расширяющаяся. Сжимающаяся часть представляют обычную структуру свёрточной сети (свёртка, активация, max pooling). Причём после каждого уменьшения нейронной сети число feature map удваивается. Расширяющаяся часть состоит из up sampling слоёв и обычных свёрточных слоёв (свёртка + активация), при этом уменьшается число feature map. Последний слой – это свёрточный слой(1x1), который присваивает каждому пикселю вероятность: содержит ли он здание. Данная сеть представляет U-образный вид (отсюда она и получила своё название). Причём в процессе обучения изображение сперва уменьшается в размере, но увеличивается по количеству feature map. Затем наоборот изображение увеличивается в размере при этом число feature map уменьшается. Feature map из сжимающейся части сети пробрасываются на расширяющуюся часть.

Слои up sampling – обратные слоям максимального объединения, а именно увеличивают размер изображения, дублируя значение пикселя в области заданного размера.

Базовый блок сжимающей части нейронной сети следующий:

Convolution	3x3

Batch normalization	$x_i \leftarrow \frac{x_i - M[x]}{\sqrt{D[x] + \epsilon}}$ $y_i \leftarrow \gamma x_i + \beta$
ELU	$g(z) = \begin{cases} z, z \geq 0 \\ a * (e^z - 1) \end{cases}$
Convolution	3x3
Batch normalization	$x_i \leftarrow \frac{x_i - M[x]}{\sqrt{D[x] + \epsilon}}$ $y_i \leftarrow \gamma x_i + \beta$
ELU	$g(z) = \begin{cases} z, z \geq 0 \\ a * (e^z - 1) \end{cases}$
Max pooling	2x2

Таблица 1 – базовый блок U-Net

Сперва идёт свёртка, где размер локальной области – 3x3 пикселя. Затем следует батч нормализация [4]. В качестве функции активации используется elu (1.8). Можно использовалась и relu (1.7), однако тогда обучение происходит медленнее. В конце идёт max pooling по области 2x2 пикселя.

Нормализация является стандартным подходом и применяется для входных данных в нейронную сеть. Например, если мы работаем с изображением удобно чтобы все значения входа в сеть были в диапазоне от 0 до 1. Идея батч нормализации сделать нормализацию частью архитектуры сети и примерять для каждого батча данных. Такая нормализация обеспечивает стабильное распределение данных входных слоёв нейронной сети, по мере того как сеть тренируется, что позволяет использовать параметр скорости обучения выше и обращать меньше внимания на инициализацию весов и смещений. Также батч нормализация служит регуляризатором.

Алгоритм батч нормализации:

1. $M[x] \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ - математическое ожидание для батча
2. $D[x] \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - M[x])^2$ - дисперсия для батча
3. $x_i \leftarrow \frac{x_i - M[x]}{\sqrt{D[x]+\epsilon}}$ - нормализуем
4. $y_i \leftarrow \gamma x_i + \beta$ - вычисляем новое значение

Таким образом мы нормализуем данные перед каждой функцией активации. Параметры γ, β обучаются во время обратного распространения ошибки. Так как просто нормализация может изменить то, что данный слой должен был представлять. По сути батч нормализация – это ещё один слой нейронной сети с новыми параметрами для обучения.

Расширяющийся блок нейронной сети аналогичный за исключением, что слой max pooling заменяется на up-sampling, где происходит увеличение разрешения изображения.

В Таблица 2 показаны другие параметры, которые использовались при обучении сети.

Классификатор	$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$
Функция стоимости	$-\frac{1}{n}\sum_{x}\sum_{j}(\, y_j \ln a_j + (1-y_j)\ln(1-a_j))$
Обучение	SGD, Adam
Скорость обучения	0.001

Таблица 2 – обучение сети

В последнем слое сети для классификации значения пикселя использовался сигмоид (1.6). Так как необходимо производить бинарную классификацию он подходит лучше всего. Предоставляет в качестве выхода вероятность того, что данный пиксель содержит здание. Функция стоимости – binary cross entropy (1.4), подходит хорошо для задач классификации по причинам, описанным выше. Для обучения использовался алгоритм стохастического градиентного спуска, а также Adam [12], который по результатам экспериментов превзошёл использование SGD.

Т.е. в стандартную сеть U-Net была добавлена батч нормализция и вместо функции активации relu использовалась elu.

3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

Данная глава описывает практическую часть работы. Сперва рассмотрим набор данных, который использовался для оценки эффективности алгоритма, затем сам процесс обучения, эксперименты и трудности с которыми столкнулись в процессе обучения. Результаты и дальнейшее усовершенствование в сегментации изображений.

3.1 Данные

Данные предоставлены французским университетом и подробно описаны в [11]. Тренировочное множество представлено 180 изображениями. Обычные трёхканальные RGB изображения размером 5000x5000 пикселей. Среди изображений присутствуют снимки сельской местности и городской застройки. Они включают в себя разные города США и Австрии. Всего 5 городов по 36 снимков на каждый: Остин, Чикаго, Китсап, Западный Тироль, Вена.

Кроме снимков городов тестовое множество содержит маски для каждого снимка: чёрно-белые изображения, где здания выделены белым цветом всё остальное – чёрный. Задача сводится к бинарной классификации каждого пикселя (возможны только два класса: здание и не здание).

На Рисунок 3.1 представлено несколько изображений.

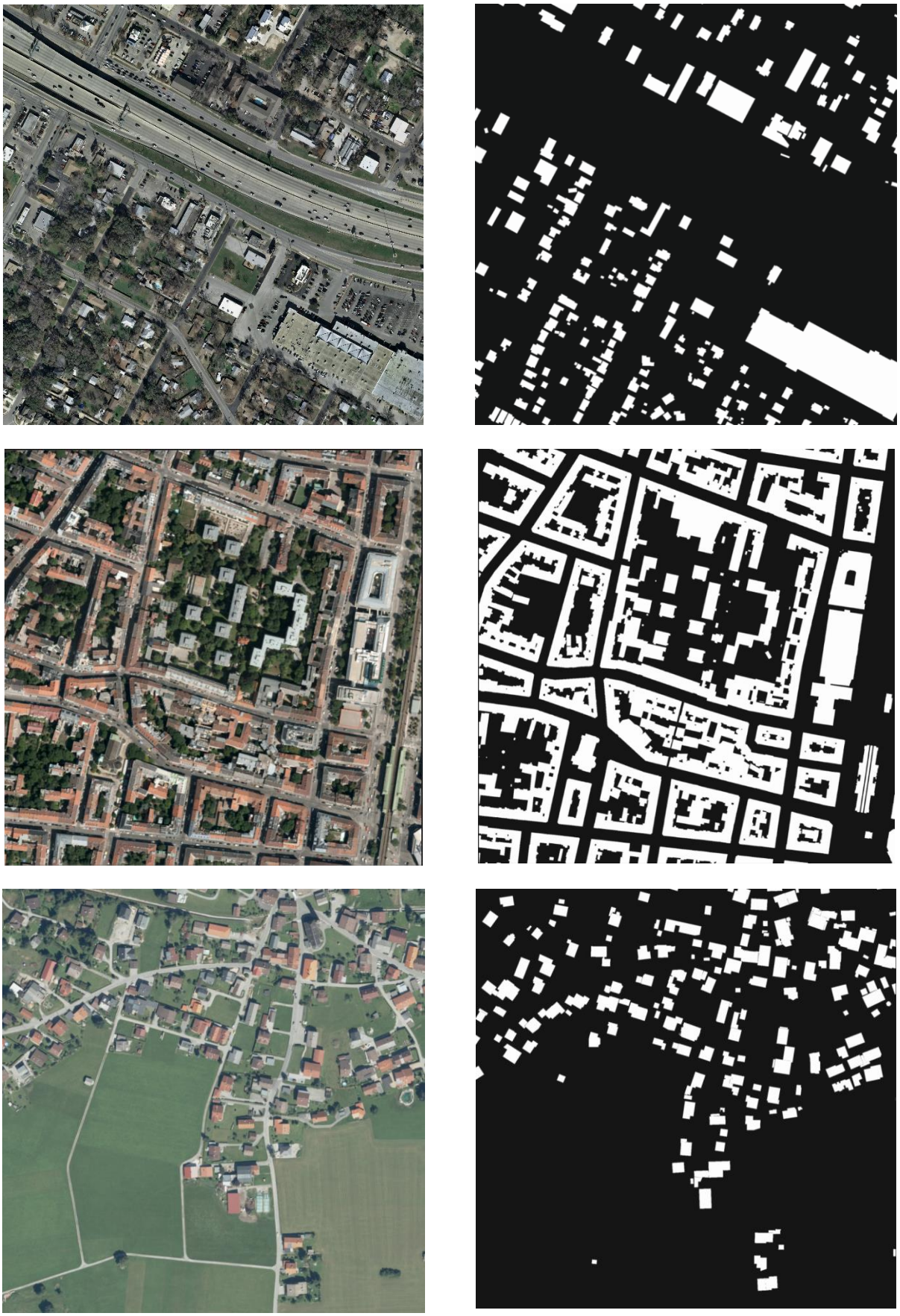

Рисунок 3.1

Проверочное тестовое множество также содержит 180 изображений размера 5000x5000. Причём это снимки других городов США и Австрии, которые отсутствуют в тренировочном множестве (Беллингхем, Блумингтон, Инсбрук, Сан-Франциско, Восточный Тироль). Несколько тестовых изображений представлено на Рисунок 3.2.

Рисунок 3.2

Как мы видим тренировочное и тестовое множество представлены разными городами. Присутствуют как крупные города, так и небольшие. Снимки включают леса, поля, реки и озёра. Разные города в тестовом и тренировочном множествах помогают проверить как алгоритм может обобщаться на разных данных.

3.2 Метрики

Для оценки качества результата работы алгоритма. Использовались следующие метрики:

Точность: процент правильно обозначенных пикселей.

IoU: отношение пересечения пикселей, являющихся зданиями и выделенных как здания с помощью алгоритма, к объединению пикселей, являющихся зданиями и выделенных с помощью алгоритма.

Т.е. если A – множество пикселей со зданиями, полученных с помощью алгоритма, B – множество пикселей, действительно являющихся зданиями на изображении (ground truth), то:

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

Пример IoU для двух множеств на Рисунок 3.3:

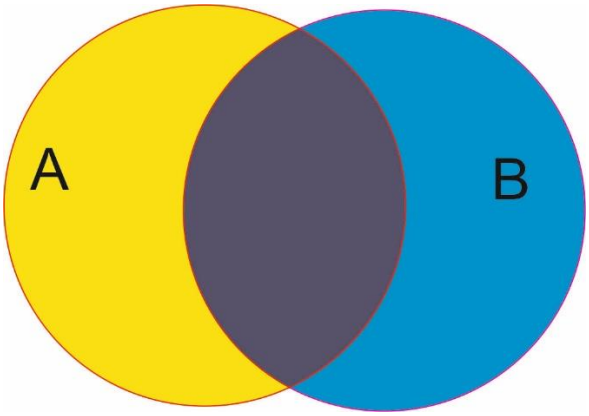

Рисунок 3.3

IoU более наглядно оценивает работу алгоритма, так как точность не обращает внимание на классы представленные небольшим числом пикселей. Например, в случае, если 90 процентов изображения составляют леса, озёра и т.д., а на оставшихся 10 процентах присутствуют здания, наш алгоритм может хорошо распознать отсутствие зданий на большой территории при этом точность будет высокая около 90 процентов, но IoU в тоже время может быть небольшим: 30 процентов (здания были выделены плохо). Если после улучшения алгоритма точность вырастет на 2-3 процента, IoU может измениться на 20-30 процентов. Т.е. изменения и значения IoU более показательны. IoU хорошо подходит для несбалансированных наборов данных, таким является и используемый в работе массив данных.

3.3 Обучение

3.3.1 Средства, используемые при обучении

Для написания приложения по сегментации зданий в данной работе использовался язык python (версия 3.6). Является одним из наиболее удобных языков при работе с математическими вычислениями. Существуют библиотеки для научных вычислений. Например, NumPy – предназначена для работы с N-мерными массивами, быстрых вычислений в области линейной алгебры и др. Существует несколько фреймворков машинного обучения для python. Один из самых популярных – keras, является высокоуровневым и достаточно простым к освоению. Он служит обёрткой над другими фреймворками машинного обучения такими как TensorFlow, Theano, CNTK. В данной работе использовался keras с бекендом TensorFlow.

Обучение нейронных сетей обычно производится на видеокартах, из-за хорошей способности распараллеливать вычисления. Все эксперименты в данной работе осуществлялись на видеокарте GTX 1050 с 4 ГБ памяти.

3.3.2 Подготовка данных

Как говорилось ранее исходные данные представляют собой изображения размером 5000x5000 пикселей. Такие изображения являются слишком большими, чтобы подавать их на вход нейронной сети. Поэтому из исходных изображений случайным образом нарезались группы изображений меньшей размерности. Размер изображения коррелирует с размером мини батча, т.е. чем больше размер изображения, тем меньше можно будет выбрать мини батч для тренировки и наоборот. Также важно на сколько удалённым является снимок земли и сколько зданий находится на нём. Параметры для размера изображения и мини батча подбирались экспериментально. Размер изображения был выбран 224x224. Максимально возможный размер батча оказался равным 8. Из каждого изображения случайным образом вырезалось некоторое количество изображений 224x224. Одна седьмая часть этих изображений была для валидации, остальные помещались в тренировочное множество. Для предсказания тестовые изображения также нарезаются под размер входа сети. Для каждого нарезанного изображения предсказывается бинарная маска, затем все предсказанные маски склеиваются в одну общую для исходного тестового изображения. Так как размер тестового изображения не является кратным нарезаемому сегменту, вокруг него строится симметричная рамка Рисунок 3.8.

3.3.3 Результаты экспериментов

Первый шаг

Для первого эксперимента было использовано около 8000 изображений. Результат для снимка из тестового множества представлен на Рисунок 3.4.

Рисунок 3.4

Видно, что результатом несколько часового обучения является пустая картинка, хотя во время обучения метрики на валидационном множестве

показывали удовлетворительные результаты. Как оказалось изначальное распределение данных было плохим.

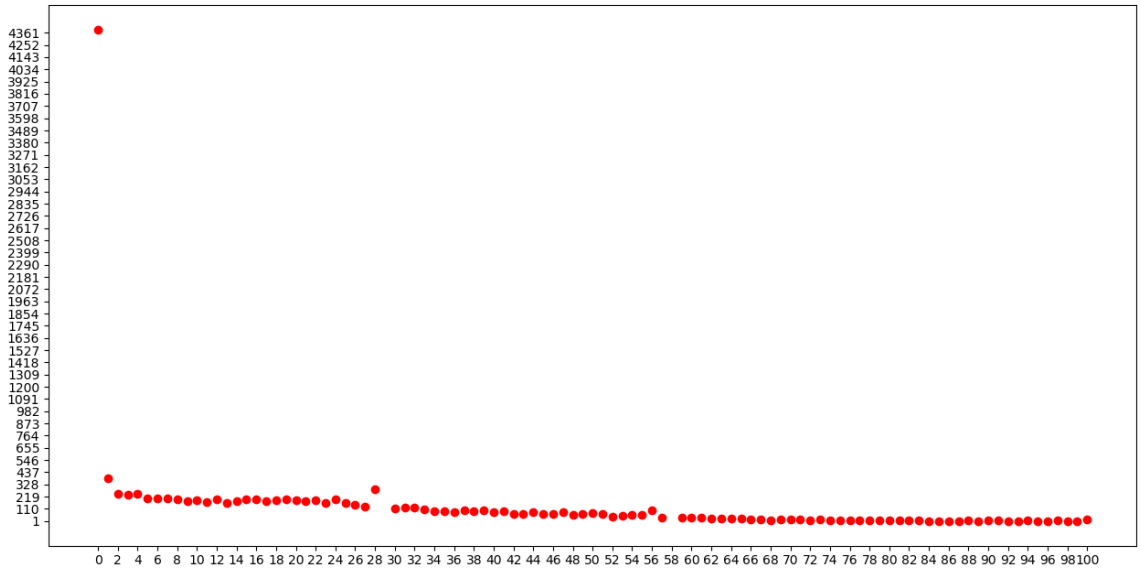

Рисунок 3.5

На Рисунок 3.4 показано какой процент зданий на скольких изображениях присутствует. Половина изображений оказалась пустой! Нейронная сеть обучилась на таких изображениях, нашла некоторый, удобный локальный минимум и для любого изображения из тестового множества предсказывало пустое изображение.

Второй шаг

Вводится порог на присутствие зданий на изображении. Минимум 10% изображения должны составлять здания. На этом этапе в сеть также добавляется батч нормализация. Производится расширение множества тренировочных данных. Каждое изображение и соответствующая ему маска поворачивается на 90 градусов. Таким образом число изображений увеличивается в два раза. Вообще, при недостаточном наборе данных число изображений можно расширить в 8 раз (4 поворота и отражения). Но в данной задаче недостатка в данных нет. Так как изображения достаточно большие можно просто вырезать из каждого изображения больше областей (в итоге и был применён подход без аугументации данных).

Тренировка занимает около 25 часов. Результат на Рисунок 3.6.

Рисунок 3.6

В данном случае здания уже определяются неплохо, однако нейронная сеть распознаёт воду как здания. Поэтому на следующем шаге в исходное множество также отдельно добавляются снимки с водой.

Третий шаг

Итоговое распределение зданий на изображениях представлено на Рисунок 3.7.

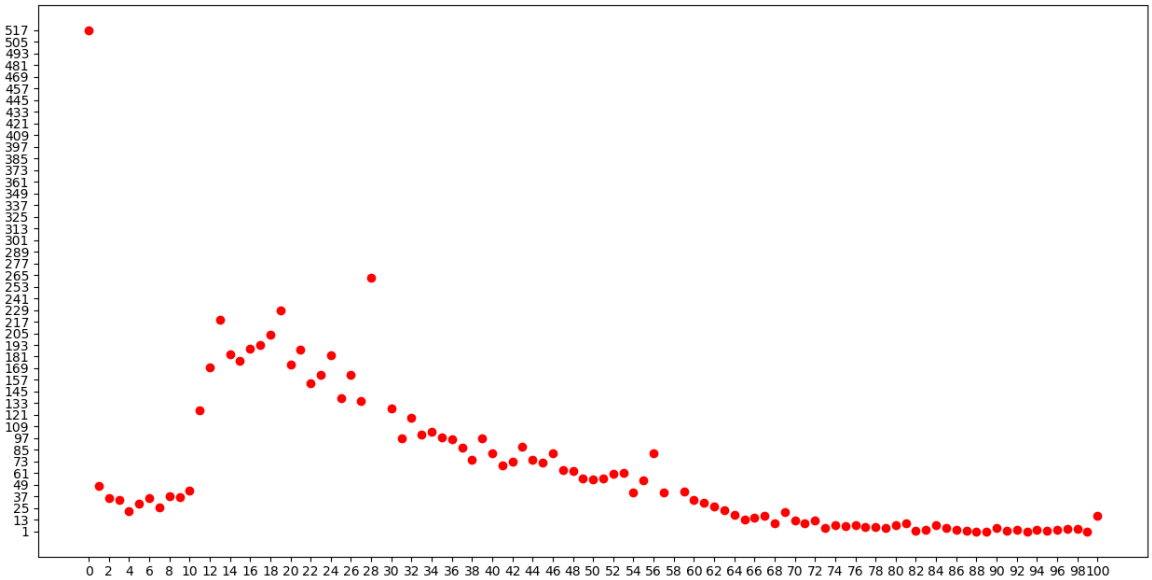

Рисунок 3.7

На этом этапе была решена проблема погрешности на краях изображения. Нейронная сеть плохо предсказывала здания по краям вырезанной области изображения. Там начинали появляться различные артефакты. Чтобы избежать этой погрешности тестовая картинка разбивается на пересекающиеся области 224x224 и для каждой выполняется предсказание, строится маска из

непересекающихся сегментов 180x180, и все такие сегменты заносятся в результат Рисунок 3.8.

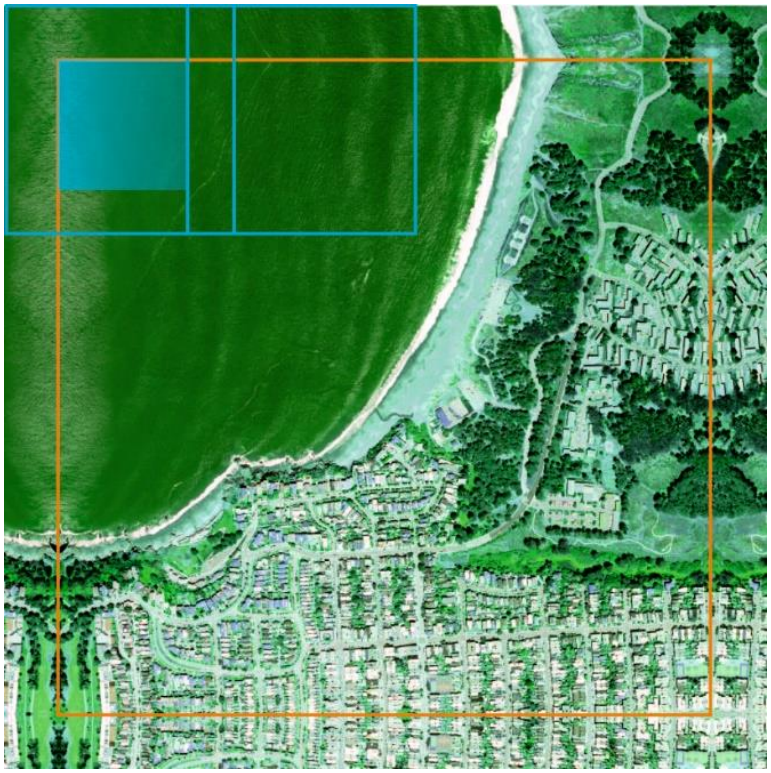

Рисунок 3.8

На этом этапе результаты работа алгоритма выглядят достаточно неплохо Рисунок 3.9.

Четвёртый шаг

Для улучшения эффективности обучения также использовалась кросс-валидация. Исходное множество данных разбивается на несколько долей, например, на 3, обучается на двух тестируется на третьей, затем обучается на двух других. Помогает определить переобучение на этапе тренировки сети. Однако в данном эксперименте кросс-валидация не принесла конкретной выгоды, только увеличилась продолжительность процесса обучения.

Также для предсказания использовалась аугументация данных во время тестирования (test time augmentation). Предсказания выполняются по нескольким выходам нейронной сети, для изображения и для повернутого изображения на 90 градусов. Такой способ заметно уменьшает число False Positive (Объекты, воспринятые как здания, но ими не являющиеся). Однако на практике оказалось вместе с уменьшением ложных объектов, отбрасывались и правильно выделенные знания. Поэтому результирующая сеть работает без test time augmentation.

Рисунок 3.9

Рассмотрим, как с изменением подхода улучшились метрики:

	Беллингхем	Блумингтон	Инсбрук	Сан-Франциско	Тироль
Точность	91.74	83.43	90.71	87.47	90.80
IoU	43.12	25.92	46.29	65.93	40.03

Таблица 3 – метрики по городам, один из первых результатов

	Беллингхем	Блумингтон	Инсбрук	Сан-Франциско	Тироль
Точность	96.14	94.69	95.79	90.53	97.06
IoU	61.29	51.18	63.64	72.07	67.56

Таблица 4 – лучшие метрики по городам

Видно, что последний результата значительно превосходит первые попытки.

3.4 Обсуждение, сравнение с известными результатами

В данной работе продемонстрирован подход с использованием глубоких нейронных сетей для сегментации зданий на спутниковых изображениях.

В ходе экспериментов было выяснено насколько важно правильно подбирать тренировочное множество. Одни и те же изображения можно представить по-разному. Не должны преобладать пустые изображения или изображения, полностью заполненные строениями. Правильное использование тренировочного множества помогает значительно улучшить результат. Сперва необходимо изучить данные. Стоит использовать те изображения, которые обладают необходимыми свойствами и нужно опасаться значительного преобладания конкретного свойства в тренировочном множестве.

Модификации, которые были использованы для U-Net известные и приносят значительное улучшение в качестве работы сети.

Была решена проблема с погрешностью на краях изображения с помощью предсказания внутренней части сегмента.

Следует отметить, что результат данной работы превзошёл оригинальной результат, описанный в [11] (работа предоставляющая исходный массив данных). В Таблица 5 приведены метрики оригинальной работы. Однако всё равно полученный результат уступает результатам других специалистов, которые занимались выделением зданий на данном массиве изображений.

	Беллингхем	Блумингтон	Инсбрук	Сан-Франциско	Тироль
Точность	95.37	95.27	95.37	87.00	96.61
IoU	56.11	50.40	61.03	61.38	62.51

Таблица 5 – метрики, полученные в оригинальной работе

В связи с этим следует заметить, что у алгоритма существуют дальнейшие улучшения. Можно попробовать использовать ансамбль сетей. Т.е. предсказывать значение каждого пикселя не согласно выхода из одной сети, а в соответствии с выходом нескольких сетей. Результат предсказания можно вычислять как среднее или каким-то другим способом. В качестве дополнительной сети можно использовать свёрточную глубокую нейронную сеть tiramisu [13]. Для выделения воды можно попробовать использовать индексы и совмещать с результатом сети. Следует также обратить большее внимание на границы зданий и правильное разделение рядом стоящих объектов.

ЗАКЛЮЧЕНИЕ

Данная работа рассматривает алгоритмы, использующие нейронные сети для выделения зданий на изображениях. Были изучены существующие подходы и алгоритмы сегментации изображений с помощью нейронных сетей. Для решения поставленной задачи была реализована нейронная сеть U-Net. Проведена оценка эффективности и сравнение с другими известными результатами. В конце работы рассмотрены дальнейшие пути развития и возможное улучшение.

Результатом работы является приложение способное выделять здания на спутниковых изображениях на основе обученных моделей. Необходимо подготовить тренировочное множество. Далее приложение позволяет обучить сеть, и обученная модель будет распознавать здания. В работе было обращено внимание на распределение данных в обучающем множестве. Очень важно подобрать хороший набор тренировочных данных.

Важно заметить, что использованный подход и нейронную сеть можно использовать не только для выделения зданий, но и для других объектов на снимке (озёра, реки). Главное предоставить необходимое обучающее множество.

ПРИЛОЖЕНИЕ А.

Репозиторий с приложением: https://github.com/Alesiks/satellite-images-segmentation

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Nielsen, A. M. Neural Networks and Deep Learning / A. M. Nielsen – Determination Press, 2015
2. Goodfellow, I. Deep learning / I. Goodfellow, Y. Bengio, A. Courville. – MIT Press, 2016 – 785p.
3. Ciresan, D. C. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images / D. C. Ciresan, A. Giust, L. G. Gambardella, J. Schmidhuber, – NIPS, 2012 – 9p.
4. Ioffe, S. Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift / S. Ioffe, C. Szegedy. – arXiv preprint, 2015 – 11p.
5. Clevert, D. Fast and accurate deep network learning by exponential linear units (Elus) / D. Clevert, T. Unterthiner, S. Hochreiter, – arXiv preprint, 2016 – 14p.
6. Krizhevsky, A. Imagenet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. E. Hinton, – NIPS, 2012 – 9p.
7. Sırmacek, B. Building Detection from Aerial Images using Invariant Color Features and Shadow Information / B. Sırmacek, C. Unsalan. – 23rd International Symposium on Computer and Information Sciences 2008 – 5p.
8. Chartock, E. Extraction of Building Footprints from Satellite Imagery / E. Chartock, W. LaRow, V. Singh – 2017 – 8p.
9. Iglovikov, V. Satellite Imagery Feature Detection using Deep Convolutional Neural Network: A Kaggle Competition / V. Iglovikov, S. Mushinskiy, V. Osin. – arXiv preprint, 2017 – 6p.
10. Ronneberger, O., Fischer, P., Brox, T, U-Net: Convolutional Networks for Biomedical Image Segmentation / O. Ronneberger, P. Fischer, T. Brox. – Springer International Publishing 2015 – 8p.
11. Maggiori, E., Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark / E. Maggiori, Y. Tarabalka, G. Charpiat, P. Alliez. – IEEE International Symposium on Geoscience and Remote Sensin, 2017 – 5p.
12. Kingma, P., D., ADAM: A method for stochastic optimization / P. D. Kingma, J. Ba. – arXiv preprint 2015 – 15p.
13. Jégou, S. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation/ S. Jégoum M. Drozdzal, D. Vazquez, A. Romero, , Y. Bengio – arXiv preprint 2016 – 9p.
14. Dstl Satellite Imagery Feature Detection. [Electronic resource]. – Mode of access: https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection – Date of access: 20.12.2017

15. Keras. [Electronic resource]. – Mode of access: https://keras.io/ – Date of access: 30.05.2018
16. Inria Aerial Image Labeling Dataset. [Electronic resource]. – Mode of access: https://project.inria.fr/aerialimagelabeling/contest/ – Date of access: 30.05.2018