Министерство образования и науки Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Иркутский государственный университет»
(ФГБОУ ВО «ИГУ»)
Институт математики, экономики и информатики
Кафедра вычислительной математики и оптимизации

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА

по направлению «прикладная математика и информатика»
профиль «математическое и компьютерное моделирование»

ПРИЛОЖЕНИЕ КОНКУРИРУЮЩИХ СЕТЕЙ ДЛЯ КОЛОРИЗАЦИИ ИЗОБРАЖЕНИЙ

Студента 4 курса очного отделения
группы 02421-дб
Семёнова Тимофея Алексеевича

Руководитель:
д. ф.-м. н., профессор
_____Сидоров Д. Н.

Допущена к защите
Зав. кафедрой, д. ф.-м. н, профессор
_____Дыхта В. А.

Иркутск - 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цифровая обработка изображений всегда была одной из самых сложных областей в компьютерных науках. Начав зарождаться как часть области обработки сигналов, обработка изображений постепенно стала самостоятельным направлением, называемым компьютерным зрением. В 1966 году исследователи считали, что одного летнего проекта для группы студентов [19] будет достаточно, чтобы закрыть многие вопросы компьютерного зрения, но в итоге работы в этом направлении ведутся и по сей день.

Одной из интригующих задач компьютерного зрения является задача восстановление цветных изображений. Долгое время многие подходы к этой задаче требовали очень тщательного анализа изображений и придумывания способов извлечения полезных признаков [2, 18]. С появлением больших вычислительных мощностей и бурным развитием искусственных нейронных сетей стало возможным использование нейросетевых подходов ко многим задачам компьютерного зрения [17], включая задачу колоризации изображений [12]. В частности стало возможным получение фотореалистичных изображений без какой-либо постобработки получаемых алгоритмами изображений.

В рамках данной работы предлагается архитектура модели конкурирующих сетей для полностью автоматического раскрашивания черно-белых изображений. Также приводится необходимая информация и теория об использовании и обучении нейронных сетей. Описанный метод позволяет уйти от ручного анализа данных и придумывания полезных признаков.

Глава 1. Задача колоризации изображения

1.1. Общая постановка задачи

В общем виде модель колоризации изображения может быть сформулирована в виде:

$$y = \phi(x, t)$$

Здесь $t \in \mathbb{T}, x \in \mathbb{R}^{h \times w \times c_1}, y \in \mathbb{R}^{h \times w \times c_2}, \phi : \mathbb{R}^{h \times w \times c_1} \times \mathbb{T} \to \mathbb{R}^{h \times w \times c_2}$. x – это исходное c_1-канальное изображение размера $h \times w$ с частично сохранившейся информации о цветах, t – некоторая дополнительная информация об изображении (например, информация об исходном распределении цветов), ϕ – некоторое преобразование (например, нейронная сеть), y – колоризованное c_2-канальное изображение. Ставится задача предложить такое преобразование $\phi(x, t)$, чтобы получаемые цветные изображения были как можно более *реалистичные*. Заметим, что преобразование $\phi(x, t)$ может быть определенно неоднозначно, так как различным цветным изображениям y может соответствовать один и тот же x, поэтому под *реалистичностью* изображения y понимается не соответствие оригиналу изображения x (которого может и не существовать), а визуальное восприятие человеком.

1.2. Обзор существующих методов решения

Методы для решения данной задачи различаются как начальными предположениями о преобразовании ϕ, так и наличием или отсутствием той или иной информации об исходном изображении. Условно разделим существующие методы на 3 группы:

1) Методы переноса стиля.

2) Методы распространения цветов.

3) Методы без дополнительной информации.

Методы переноса стиля

Методы данной группы используют в качестве t некоторое дополнительное цветное изображение, чей *стиль* переноситься на исходное черно-белое изображение x. При классическом подходе под *стилем* понимается некоторое соответствие между пикселями изображений x и t, определяемое с помощью локальных дескрипторов, как, например, использование дескриптора SURF [23] в работе G.Charpiat, *et al* [3]. Предложенные методы зачастую страдают от наличия множественных артефактов: слияние объектов с фоном, резкие переходы не соответствующие текстуре, блеклые цвета. Для борьбы с дефектами используются разные приёмы пост-обработки, например, предложенная в работе A.Bugeu, *et al* [2] регуляризация с помощью метода тотальной вариации.

1 2 3 4 5

Рисунок 1. 1 – источник *стиля*, 2 – исходное изображение, 3, 4, 5 – результаты работы классических алгоритмов переноса стиля [2].

В последние годы значительные результаты были получены с использованием нейронных сетей [6] – для них *стиль* определяется как матрица Грама, полученная из линеаризованных выходов некоторого слоя нейронной сети. Преимуществом такого подхода является возможность переносить *стиль* на любое изображение, не обязательно черно-белое.

Методы распространения цветов

Методы данной группы предполагают взаимодействие с пользователем, который должен сделать наброски желаемых цветов на исходном изображении – для удобства, можно считать, что вся необходимая информация есть в исходном изображении x, а t отсутствует. Для того, чтобы раскрасить недостающие участки, минимизируется некоторый функционал, который штрафует за разницу в цвете между соседними пикселями со схожей интенсивностью [22, 18]. Данные методы позволяют пользователю контролировать получаемый результат и итеративно улучшать качество колоризации.

Методы без дополнительной информации

Данная группа методов не использует никакой дополнительной информации об исходном изображении x. В данном подходе используются методы машинного обучения для решения некоторой обратной задачи: пусть имеется тренировочная выборка изображений D_{train} из генеральной совокупности цветных изображений D_{real}, $\Phi = \{\phi(\cdot) : \phi(x) = \phi(x, \theta), x \in D_{real}, \theta \in \Theta\}$ – некоторое параметрическое семейство функций, $L : (D_{train}, \phi) \to \mathbb{R}$ – некоторый функционал качества, который необходим минимизировать по всем $\phi \in \Phi$. Зачастую в качестве функционала качества L используется сумма по всем тренировочным изображениям среднеквадратичной ошибки между оригинальным цветным изображением и колоризованным. Однако полученные при таком подходе восстановленные изображение склонны иметь ненасыщенные и тусклые цвета [11] – это может быть связано с тем, что многие изображения (как следствие изображения из тренировочной выборки) содержат какой-то неяркий фон (облака, земля, вода, и т.п.). Для улучшения визуального качества можно сначала получать распределение цветов для пикселя, а уже потом делать точечное предсказание [26], но это всё ещё далеко от того, чтобы работать с функционалом

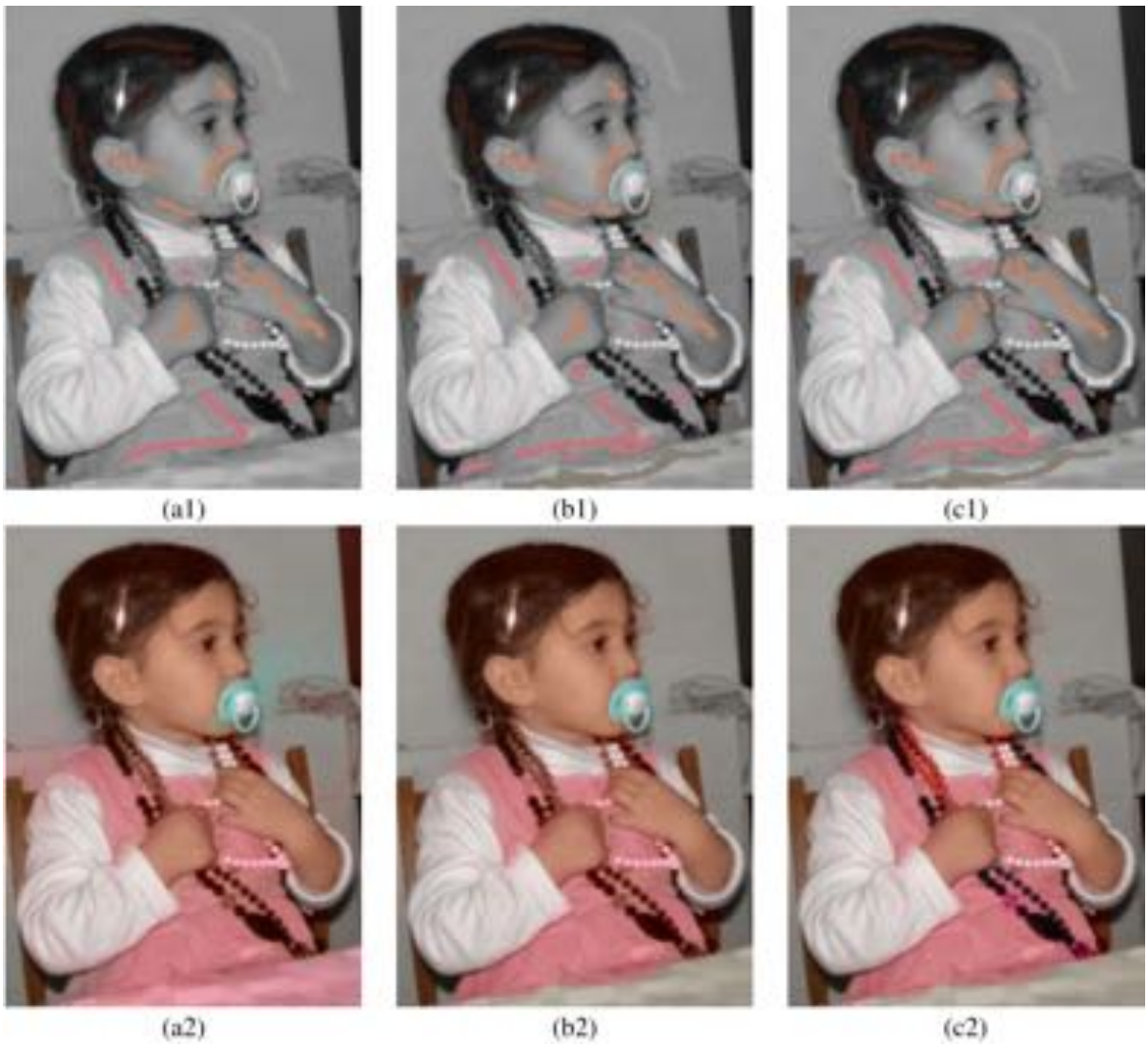


Рисунок 2. Постепенное улучшение колоризации [18].

качества, который бы соответствовал тому, как человек воспринимает изображения. В рамках данной работы проводится исследование возможностей улучшения существующих нейросетевых подходов к решению задачи колоризации изображений. Для этого была использована недавно предложенная модель конкурирующих сетей [7], способная автоматически находить функционал качества отвечающий за визуальное восприятие получаемых изображений.

Глава 2. Модель конкурирующих сетей

Область нейронных сетей изначально была мотивирована целью смоделировать работу биологических нейронных систем, но с тех пор отошла от этого направления и сейчас является одним из методов машинного обучения. Одним из первых в этой области получил результаты F.Rosenblatt, создавший модель перцептрона в 1958 году. С тех пор нейронные сети получили большое развитие благодаря развитию вычислительных мощностей и улучшению наборов данных.

2.1. Нейронные сети

В общем виде нейронная сеть представляет из себя композицию дифференцируемых функций. Пользуясь правилом дифференцирования сложной функции, можно использовать градиентные методы для того, чтобы оптимизировать нейронную сеть под заданную выборку тренировочных данных. Процесс обновления весов нейронной сети называется обратным распространением ошибки. На практике нейронные сети могут содержать десятки миллионов параметров, а в обучающей выборке могут быть миллионы объектов, из-за чего подсчёт градиента нейронной сети по всей выборке будет слишком сложным. В этом случае используются стохастические методы, такие как RMSprop (Root Mean Square Propagation) [25] и Adam [16].

Алгоритм 1 Алгоритм стохастической оптимизации RMSProp.

α: Размера шага.

γ: Коэффициент экспоненциального затухания.

$f(\theta)$: Стохастическая целевая функция с параметрами θ.

θ_0: Начальный вектор параметров.

$r_0 \leftarrow 0$ (Инициализация скользящего среднего).

$t \leftarrow 0$ (Инициализация номера итерации).

while параметры θ_t не сошлись **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$r_t \leftarrow \gamma \cdot r_{t-1} + (1 - \gamma) \cdot g_t^2$

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t / (\sqrt{r_t} + \epsilon)$

end while

Алгоритм 2 Алгоритм стохастической оптимизации Adam.

α: Размера шага.

$\beta_1, \beta_2 \in [0, 1)$: Коэффициенты экспоненциального затухания для оценок моментов.

$f(\theta)$: Стохастическая целевая функция с параметрами θ.

θ_0: Начальный вектор параметров.

$m_0 \leftarrow 0$ (Инициализация вектора первого момента).

$v_0 \leftarrow 0$ (Инициализация вектора второго момента).

$t \leftarrow 0$ (Инициализация номера итерации).

while параметры θ_t не сошлись **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

end while

Основополагающим для нейронных сетей является понятие нейрона:

$$y = f(\sum_i w_i x_i + b)$$

Здесь $w_i, x_i, b \in \mathbb{R}, f : \mathbb{R} \to \mathbb{R}$. x_i – это входные данные нейрона, w_i – веса нейрона, b – смещение нейрона, f – функция активации нейрона, y – выход нейрона. При выборе различных функций активации один нейрон может действовать как линейный классификатор. Наиболее часто используемые функции активации:

- Сигмоида: $\sigma(x) = 1/(1 + e^{-x})$

  Данная функция переводит действительные числа в интервал $(0, 1)$. В

частности, большие отрицательные числа становятся близки к 0, а большие положительные числа – к 1. Исторически сигмоида часто использовалась в качестве функции активации, так как её значение может быть проинтерпретировано как частота активации нейрона: 0 соответствует выключенному состоянию, а 1 соответствует полностью насыщенному. На практике сигмоида используется всё реже из-за двух своих недостатков:

1) *Сигмоида насыщается и убивает градиенты.* Нежелательно свойство сигмоиды иметь близкие к нулю градиенты в области насыщения около 0 и 1. Таким образом, если локальный градиент очень маленький, то он *убьёт* градиент, и почти никакой сигнал не сможет пройти через нейрон к его весам и далее. Также нужно аккуратно инициализировать веса нейрона, иначе большие значения могут сильно замедлить обучение сети.
2) *Выход сигмоиды не центрирован относительно нуля.* Это может сказаться на динамике градиентного спуска, так как если входные данные нейрона всегда положительны, то все элементы градиента весов w во время обратного распространения ошибки будут иметь один и тот же знак. Это может привести к нежелательному зигзагообразному поведению градиента.

- Гиперболический тангенс: $tanh(x) = 2\sigma(2x) - 1$

  Данная функция переводит действительные числа в интервал $(-1, 1)$. Так же как и сигмоида, гиперболический тангенс насыщается, но зато его выход центрирован относительно нуля.

- ReLU: $max(0, x)$

  Данная функция имеет как свои плюсы:

  1) Экспериментально было получено, что ReLU значительно ускоряет

сходимость стохастического градиентного спуска по сравнению с сигмоидой и гиперболическим тангенсом [17]. Предположительно это связано с его линейной не насыщаемой формой.

2) По сравнению с сигмоидой и гиперболическим тангенсом, которые требуют сложных вычислений (экспонента), ReLU может быть реализовано как простая пороговая функция.

так и минусы:

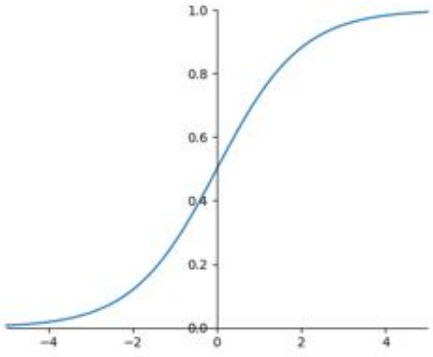
1) К сожалению ReLU может быть неустойчива во время обучения и может *умереть*. Например, очень большой градиент проходящий через нейрон может изменить веса так, что нейрон больше никогда не активируется. Если это случится, то через этот нейрон перестанет распространяться градиент ошибки.

2) Формально ReLU не дифференцируема в нуле. Обычно этим можно пренебречь в силу того, что при использовании при вычислениях чисел с плавающей запятой сложно получить точный ноль. При программной реализации значение производной в нуле зачастую полагают равной либо 0, либо 1.

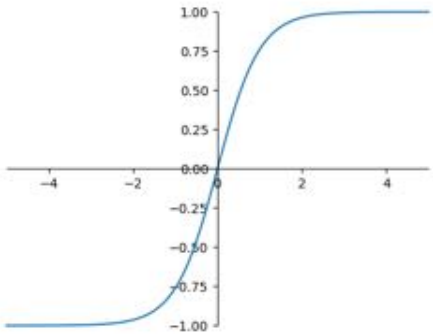
- Leaky ReLU: $max(\alpha x, x)$

  Данная функция при маленьких значениях α, ведёт себя также, как и ReLU, при этом не позволяя нейрону *умереть*.

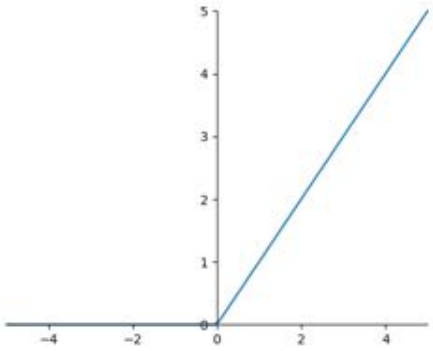
Нейронная сеть представляет из себя набор нейронов объединенных в граф без циклов. Зачастую нейроны представлены в виде последовательных слоев, соединенных друг с другом. При подсчете количества слоев в сети принято не учитывать слой с входными данными. Наиболее распространенным типом слоев является полносвязный: все нейроны текущего слоя соединены со всеми нейронами следующего слоя, но при этом не имеют между собой никаких связей.



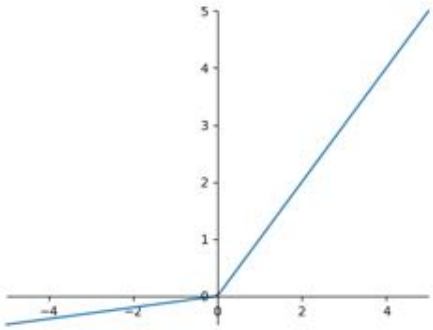
Сигмоида



Гиперболический тангенс



ReLU



Leaky ReLU

Рисунок 3. Графики различных функций активации.

В этом случае:

$$y = f(Wx + b),$$

где $x \in \mathbb{R}^n, b \in \mathbb{R}^h, W \in \mathbb{R}^h \times \mathbb{R}^n$. x – это вектор входных данных слоя, W – веса слоя, b – смещение нейрона, y – выход слоя, f – функция активации, применяемая поэлементно.

На вопрос о репрезентативной силе нейронных сетей отвечает теорема об универсальной аппроксимации [10]:

Теорема об универсальной аппроксимации

Обозначим $\phi(\cdot)$ – нетривиальную, ограниченную и монотонно-возрастающую непрерывную функцию. Обозначим I_{m_0} – m_0-мерный гиперкуб $[0,1]^{m_0}$. Пространство непрерывных функций на I_{m_0} обозначим $C(I_{m_0})$. Тогда, для любой функции $f \in C(I_{m_0})$ и любого $\epsilon > 0$, существует целое m_1 и действительные α_i, b_i и ω_{ij}, где $i = 1, \ldots, m_1$ и $j = 1, \ldots, m_0$ такие, что функция:

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \phi(\sum_{j=1}^{m_0} \omega_{ij} x_j + b_i),$$

есть аппроксимация реализации функции $f(\cdot)$, такая, что

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \epsilon,$$

для всех $x_1, x_2, \ldots, x_{m_0}$ из любой конечной выборки $D =$
$= [x_1, x_2, \ldots, x_{m_0}, f(x_1, \ldots, x_{m_0})]$.

Согласно этой теореме нейронной сети с одним скрытым слоем размера m_0 и выходным слоем размера m_1 и сигмоидными функциями активации достаточно для аппроксимации произвольной выборки с любой точностью. Однако эта теорема говорит только об аппроксимации и ничего не говорит о конкретном размере скрытого слоя m_0.

2.2. Сверточные нейронные сети

Сверточные нейронные сети весьма схожи с обычными нейронными сетями: они также состоят из нейронов, которые имеют обучаемые веса, а вся сеть по-прежнему представляет сложную дифференцируемую функцию. Главное отличие заключается в том, что архитектура сверточной нейронной сети явным образом предполагает наличие в данных локальных связей (например, зависимость пикселя от его соседей). Это позволяет сделать сеть более эффективной, и значительно снизить количество настраиваемы в ней параметров.

Сверточная нейронная сеть представляет из себя последовательность слоев, где каждый слой преобразует один объём активаций в другой с помощью дифференцируемой функции. Различают несколько типов слоёв:

- Сверточный слой.

  Сверточные слои являются ключевыми строительными блоками для сверточных нейронных сетей. Название слоя произошло от названия математической операции свертки. Пусть $f, g : \mathbb{R}^d \to \mathbb{R}$ – две функции, интегрируемые относительно меры Лебега на пространстве $\mathbb{R}^d$. Тогда их свёрткой называется функция $f * g : \mathbb{R}^d \to \mathbb{R}$, определенная формулой:

  $$(f * g)(x) = \int_{\mathbb{R}^d} f(y)g(x - y)dy$$

  В дискретном случае операцию свертки можно представить как $(f*g)(x) = \sum_t f(t)g(x - t)$. В терминах сверточных нейронных сетей f – вход слоя, обозначаемый далее I, g – фильтр, обозначаемый за K. В большинстве библиотек машинного обучения реализуется чуть другая операция – кросс-корреляция:

  $$(I * K)(x) = \sum_t I(x + t)K(x)$$

  Во время прохода через сеть мы *скользим* каждым фильтром по ширине и длине входного объёма активаций и считаем в промежуточных позициях свертку с этим фильтром (подсчет идет не только по ширине и длине, но также и по глубине). Результатом такой операции будет двумерная карта активаций, в которой записаны *отклики* фильтра на всех пространственных позициях. Каждый сверточный слой представляет из себя набор таких обучаемых фильтров, которые имеют одинаковые размерности. Интуитивно, нейронная сеть будет учить на первых слоях фильтры, которые будут активироваться при наличии тех или иных визуальных признаков,

как например края или некоторая ориентация какого-то цветового пятна, что подтверждается экспериментальными данными [17]. Укладывая карты активации разных фильтров вдоль измерения глубины, мы получим выходной объем активаций.

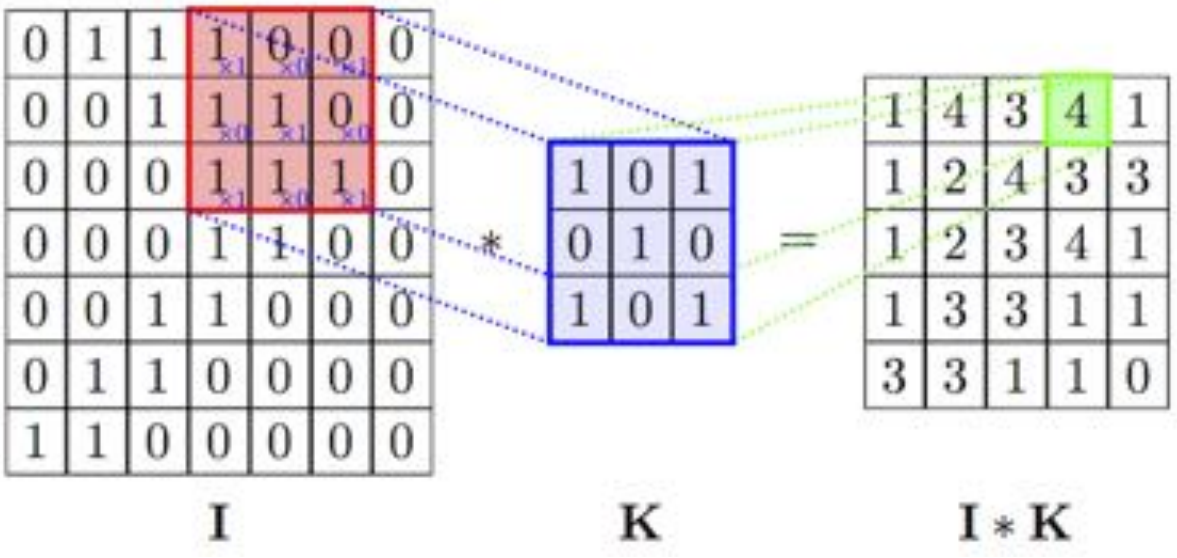


Рисунок 4. Схема вычисления свертки.

Таким образом основные отличия сверточного слоя от обычного:

– Локальная связность. В сверточном слое все нейроны соединены только с небольшим участком входного объема активаций. Пространственные масштабы такого соединения являются гиперпараметром сети и называются **рецептивным полем** нейрона (эквивалентно размеру фильтра). Важно, что степень соединения вдоль измерения глубины обязательно равно глубине всего входного объема активаций.

– Пространственное расположение. Нейроны сверточного слоя упорядочены в пространстве, а их количество регулируется с помощью **глубины** (depth), **пространственного размера** (spatial extent), **шага** (stride) и **заполнения нулями** (zero-padding). **Глубина** соответствует количеству фильтров в слое, **пространственный размер** зада-

ет ширину и высоту фильтра, **шаг** задает шаг, с которым *скользит* каждый фильтр, а **заполнение нулями** позволяет контролировать размеры карты активаций, дополняя входной объём нулями вокруг его границ.

– Совместные параметры. Благодаря тому, что один фильтр фактически соответствует большому числу нейронов, значительно сокращается количество параметров нейронной сети. Для этого используется предположение о том, что если признак, посчитанный в одной позиции, был полезен, то он также будет полезен и будучи посчитанным в другой позиции.

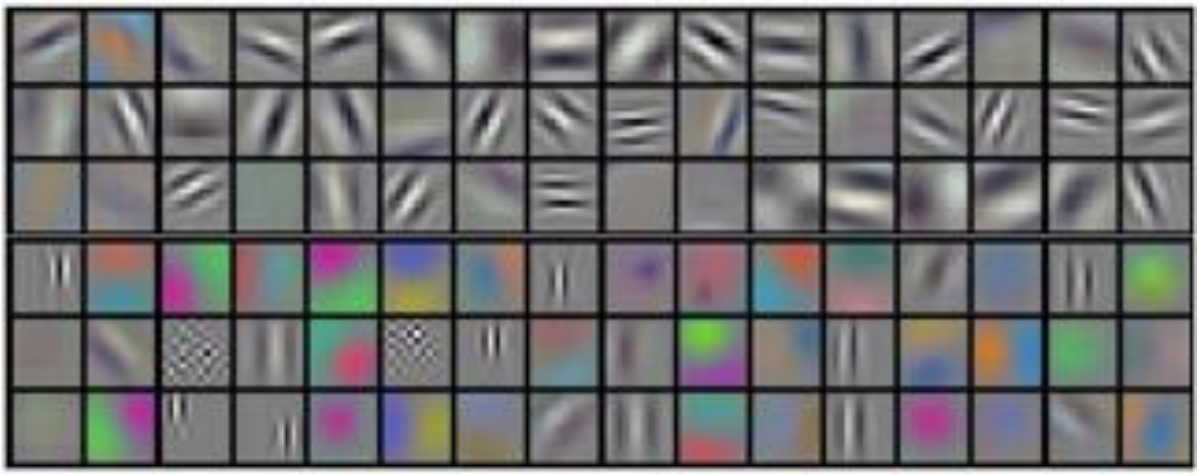


Рисунок 5. Визуализация обученных фильтров [17]. Каждый фильтр имеет размер [11×11×3] и соответствует 55*55 нейронам в одном глубинном срезе.

- Слой пулинга.

  Распространенной практикой является периодическая вставка пулинг слоев между последовательными сверточными слоями. Его задачей является постепенное уменьшение пространственных размеров внутреннего представления изображения в сети, чтобы уменьшить количество параметров и вычислений в последующих слоях, а также для контроля переобучения.

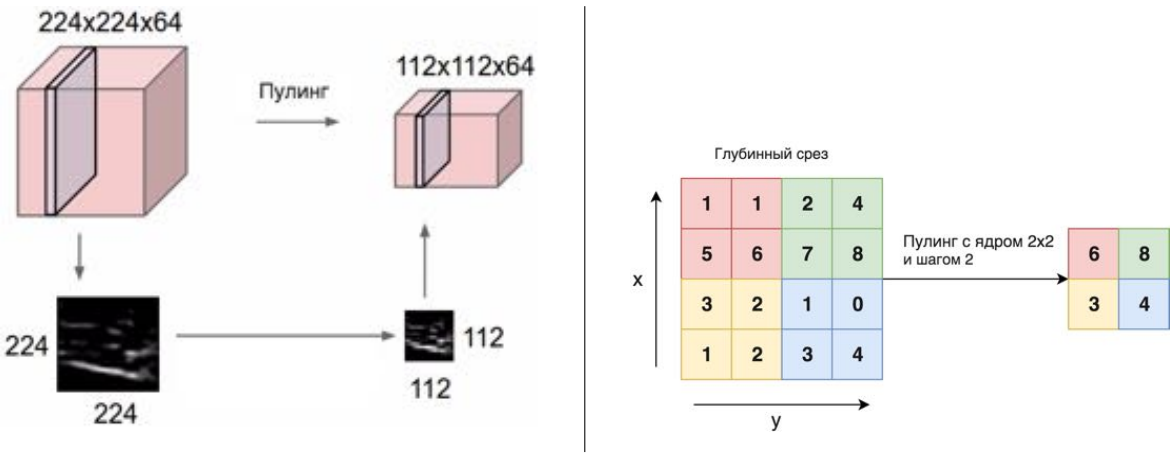


Рисунок 6. Схема работы пулинга.

Слой пулинга работает независимо на каждом глубинном срезе входа и изменяет его пространственные размеры, используя операцию взятия максимума. В качестве параметров можно задать **пространственный размер** и **шаг** слоя, аналогичные таковым у сверточного слоя.

- Полносвязный слой.

  Данный тип слоя определяется аналогично тому, как он представлен в обычных нейронных сетях. На практике последние несколько слоев нейронной сети обычно являются полносвязными [17].

- Дополнительные.

  Архитектуры почти всех сверточных нейронных сетей собраны из слоев описанных выше. Пользуясь некоторыми предположениями об исходных данных и решая конкретную задачу, можно предложить специальные типы слоев, которые также будут полезны при соблюдение некоторых условий [4].

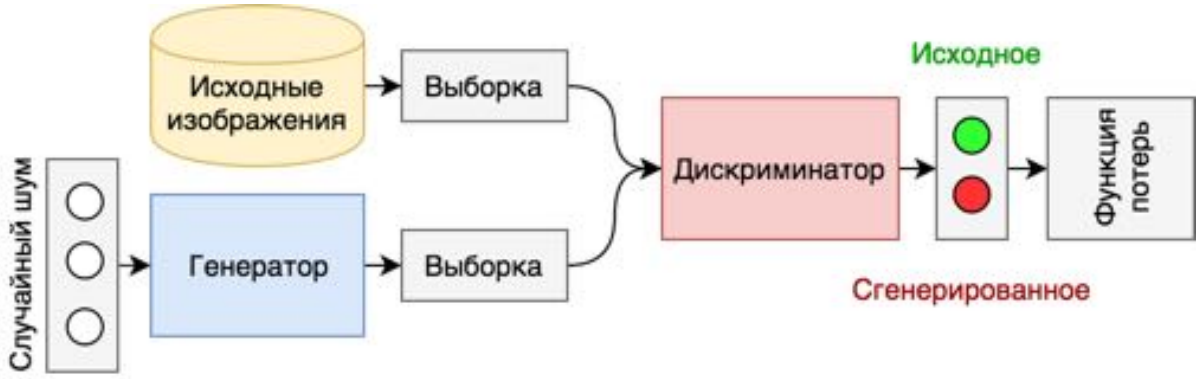


Рисунок 7. Концептуальное представление конкурирующих сетей.

2.3. Конкурирующие сети

Модель конкурирующих сетей (Generative Adversarial Networks) впервые была предложена Ian Goodfellow, *et al* [7] в 2014 году и с тех пор нашла большое применение в задачах, связанных с генерацией данных [24, 12]. Главная идея заключается в том, чтобы использовать две нейронные сети, которые будут бороться друг с другом: сеть-генератор G, получая на вход некоторый шум, выдает на выходе некоторое изображение, а сеть-дискриминатор D, получая на вход некоторое изображение, выдает на выходе вероятность того, что это изображение настоящее, а не было получено с помощью сети G. Формально модель конкурирующих сетей записывается в виде минимакс игры с функцией стоимости $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{x \sim p_z(z)}[\log(1 - D(G(z)))]$$

Здесь x – это объекты из настоящего распределения данных $p_{data}(x)$, z – некоторые модельные данные (случайный шум), получаемые из некоторого распределения $p_z(z)$. Сеть-генератор G неявно задает распределение p_g как распределение объектов $G(z)$, получаемых при $z \sim p_z$. При условии, что G и D произвольные функции, можно показать, что данная минимакс игра имеет глобальный оптимум при $p_g = p_{data}$, тем самым G будет генерировать данные идентичные

настоящим. К сожалению на практике из-за параметризации функций G и D невозможно гарантировать нахождение точного решения. Авторами также был предложен алгоритм обучения конкурирующих нейронных сетей, позволяющий достаточно сложным сетям сходиться к решению $p_g = p_{data}$.

Алгоритм 3 Стохастический градиентный спуск для обучения конкурирующих сетей. Шаг градиентного спуска может быть выполнен с помощью любого стандартного алгоритма. Авторы статьи предлагают использовать параметр $k = 1$ для уменьшения вычислительных затрат.

---

for количество итераций обучения **do**

for k шагов **do**

- Сгенерировать выборку $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ из модельного распределения $p_g(\boldsymbol{z})$.
- Сгенерировать выборку $\{\boldsymbol{x}^{(i)}\}_{i=1}^{m}$ из настоящего распределения $p_{\text{data}}(\boldsymbol{x})$.
- Обновить сеть-дискриминатор спуском по её стохастическому градиенту:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Сгенерировать выборку $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ из модельного распределения $p_g(\boldsymbol{z})$.
- Обновить сеть-генератор спуском по её стохастическому градиенту:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Обучение конкурирующих сетей является сложным и нестабильным процессом. Если сеть D будет недостаточно хорошо отличать сгенерированные данные от настоящих, то распределение p_g может смещаться в произвольную сторону, необязательно соответствующую распределению p_{data}. Если же сеть D будет обучена слишком хорошо, то она будет практически на всех объектах выда-

вать значения близкие либо к 0, либо к 1, что приведет к очень маленьким значениям градиентов и, как следствие, значительному уменьшению скорости обучения сети G. Частично эти проблемы были решены в дальнейших работах Goodfellow [13, 8], в которых были предложены важные практические рекомендации, позволяющие значительно стабилизировать процесс обучения и сделали возможным использование более глубоких моделей. Недавно был предложен новый тип конкурирующих сетей – Wasserstein GAN [1, 14] – для которых получены теоретические результаты, позволяющие избавиться от необходимости при обучении контролировать баланс между конкурирующими сетями.

Глава 3. Вычислительный эксперимент

3.1. Архитектура конкурирующих сетей

За основу взята архитектура, предложенная Phillip Isola, *et al* [12]. Пусть Ck обозначает сверточный слой с k фильтрами, использующий ReLU в качестве функции активации и BatchNorm. CDk обозначает сверточный слой с k фильтрами, использующий ReLU в качестве функции активации, и с включенным dropout с частотой 50% и BatchNorm. Все свертки имеют размер 4×4 и применяются с шагом 2.

BatchNorm

BatchNorm [15] позволяет добиться того, чтобы активации нейронов имели нормальное распределение. Стабильное распределение активаций во время обучения способствует более быстрой сходимости нейронной сети, так как слоям не приходится переучиваться работать со входами из разных распределений.

Алгоритм 4 Применение BatchNorm преобразования к активациям x. Параметры γ, β обучаются в процессе тренировки сети.

Вход: Подвыборка $B = \{x_{1...m}\}$

Выход: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

Dropout

Dropout [5] является одним из способов регуляризации нейронной сети. С вероятностью p каждый нейрон слоя может не активироваться, таким образом от этого нейрона не поступит сигнал на следующий слой.

Алгоритм 5 Применение Dropout преобразования к активациям x.

$r^{(l)} \sim Bernoulli(p)$ (Бинарная маска для слоя l)

$\tilde{y}^{(l)} = r^{(l)} * y^{(l)}$ (Применение маски к активациям слоя l)

$z_i^{(l+1)} = w_I^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}$

$y_i^{(l+1)} = f(z_i^{(l+1)})$ (Активации слоя $l+1$)

- Сеть-генератор

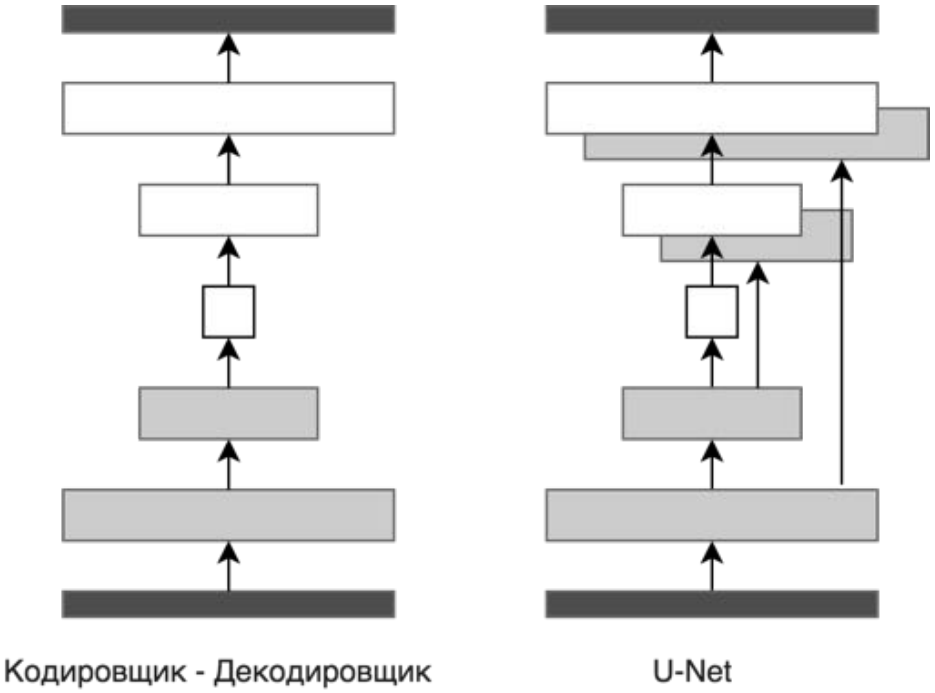


Рисунок 8. Концептуальное представление сети-генератора.

Сеть-генератор представляет из себя модификацию U-Net [21], хорошо продемонстрировавшей себя в задаче сегментации биомедицинских изображений. Архитектура сети-генератора устроена в виде *песочных часов*: выходные объемы активаций слоев постепенно снижают свою пространственную размерность, доходя до *горлышка*, а затем точно также симметрично наращивают её, тем самым на выходе сети получается изображение имеющее такие же пространственные размеры, как и входное изображение (количество каналов при этом может отличаться). Дополнительно слои,

расположенные после *горлышка*, берут на вход выходы симметричных им слоев, что позволяет одновременной обработке низко- и высокоуровневых признаков всеми слоями.

кодировщик:

$C64 - C128 - C256 - C512 - C512 - C512 - C512 - C512$

декодировщик:

$CD512 - CD512 - CD512 - C512 - C512 - C256 - C128 - C64$

После последнего слоя в декодировщике применяется свертка с двумя фильтрами (чтобы получить предсказания для ab-каналов) и функцией активации гиперболический тангенс. В качестве исключения в первом слое кодировщика $C64$ не используется BatchNorm.

- Сеть-дискриминатор

  $C64 - C128 - C256 - C512$

  В статье [12] предложена оригинальная архитектура под названием PatchGAN. Перед подачей сети-дискриминатору на вход изображения, оно разбивается на патчи размером 64×64 – таким образом для одного изображения размером 256×256 мы будем иметь 16 патчей. Каждый патч прогоняется через сеть, а затем все полученные активации подаются на вход полносвязного слоя с двумя нейронами и функцией активации софтмакс ($f(x) = \{\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}\}_{i=1}^{K}$, где K – размерность входа x). Таким образом на выходе мы получим вектор из двух элементов $[p, 1-p]$, где p – вероятность того, что входное изображение было сгенерированным.

3.2. Обучение модели

Для создания тренировочной выборки были скачаны фотографии с сервиса `http://www.flickr.com` с имеющимся тэгом "lake baikal". Полученные 14200 фотографий были случайным образом разбиты на тренировочную, валидационную и тестовую выборки в соотношении $0.7 : 0.2 : 0.1$. Затем из каждой картинки вырезается левый-верхний квадрат наибольшего размера, который масштабируется до размеров 256×256 и переводится в цветовое пространство Lab [20] – одинаковые изменения в этом цветовом пространстве производят одинаковые ощущения изменения цвета. Так как выход сети-генератора находится в пределах от -1 до 1, а цветовые каналы a и b имеют значения в пределах от 0 до 255, то во время обучения значения всех пикселей масштабируются с помощью преобразования $f(x) = \frac{x}{127.5} - 1$. Во время применения сети выполняется обратное преобразование $f^{-1}(x) = (x+1) \cdot 127.5$. Картинки подаются на вход батчами по 16 случайных изображений.

Параметры всех слоев инициализируются с помощью нормального распределения с нулевым средним и дисперсией 0.02. В работе сравниваются оригинальный метод обучения [12] и процедура WGAN, предложенная в [1].

Дополнительно к основной функции потерь сеть-генератор оптимизирует среднюю абсолютную ошибку между пикселями сгенерированного и исходного изображений для того, чтобы получаемые изображения были менее размытыми. Вся сеть обучается 10 эпох, каждая эпоха включает в себя 500 батчей.

3.3. Анализ результатов

Синим цветом на графиках представлен оригинальный метод обучения, красным – исследуемое в работе улучшение. По графика видно, что предложенный метод требует больше времени для обучения, так как сеть-дискриминатор

Алгоритм 6 WGAN. Во всех экспериментах $\alpha - 0.001, c = 0.01, m = 16, n_{critic} = 1$

w_0, θ_0: начальные параметры сети-дискриминатора и сети-генератора соответственно.

while параметры θ не сошлись **do**

for $t = 0, \ldots, n_{critic}$ **do**

Сгенерировать выборку $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ из настоящего распределения.

Сгенерировать выборку $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ из модельного распределения.

$g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$

$w \leftarrow w + \alpha \cdot RMSProp(w, g_w)$

$w \leftarrow clip(w, -c, c)$ (Зажимаем параметры в отрезок $[c, -c]$)

end for

Сгенерировать выборку $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ из модельного распределения.

$g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$

$\theta \leftarrow \theta - \alpha \cdot RMSProp(\theta, g_\theta)$

end while

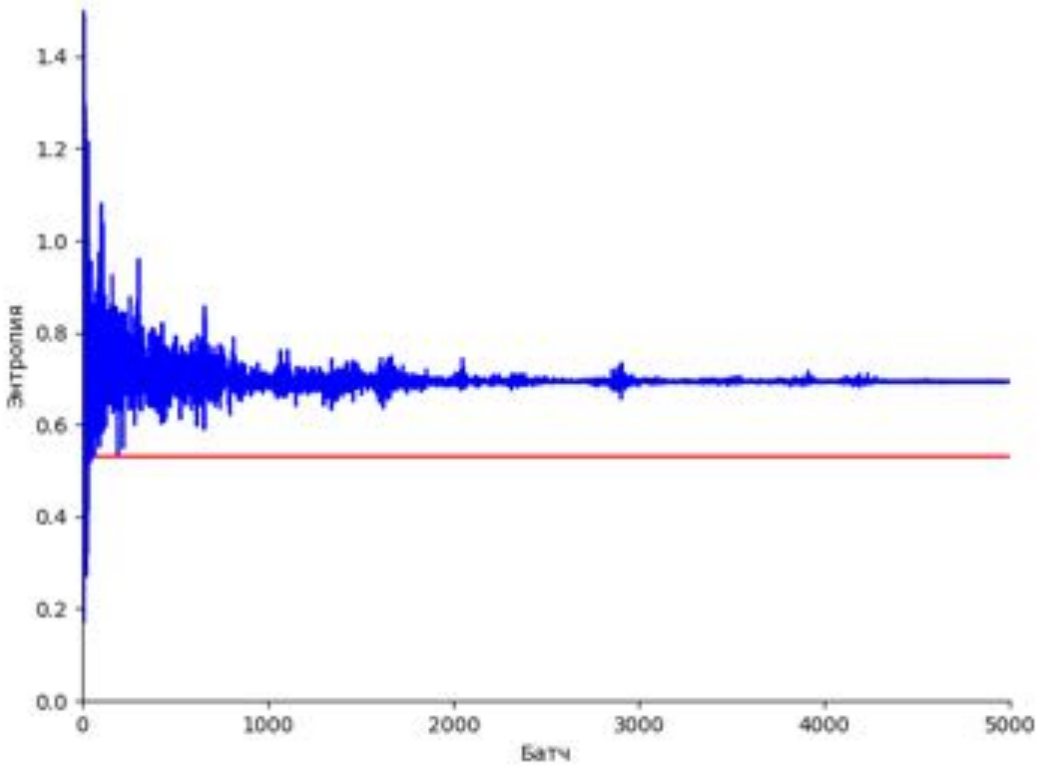


Рисунок 9. Зависимость функции потерь сети-дискриминатора от номера батча.

обновляется чаще, чем сеть-дискриминатор. Однако сеть-дискриминатор ведет себя гораздо стабильнее, а сеть-генератор обучается до того же уровня ошибки за меньшее количество батче. Это может быть критично в случае использования больших обучающих выборок, либо при использование большего разрешения для изображений.

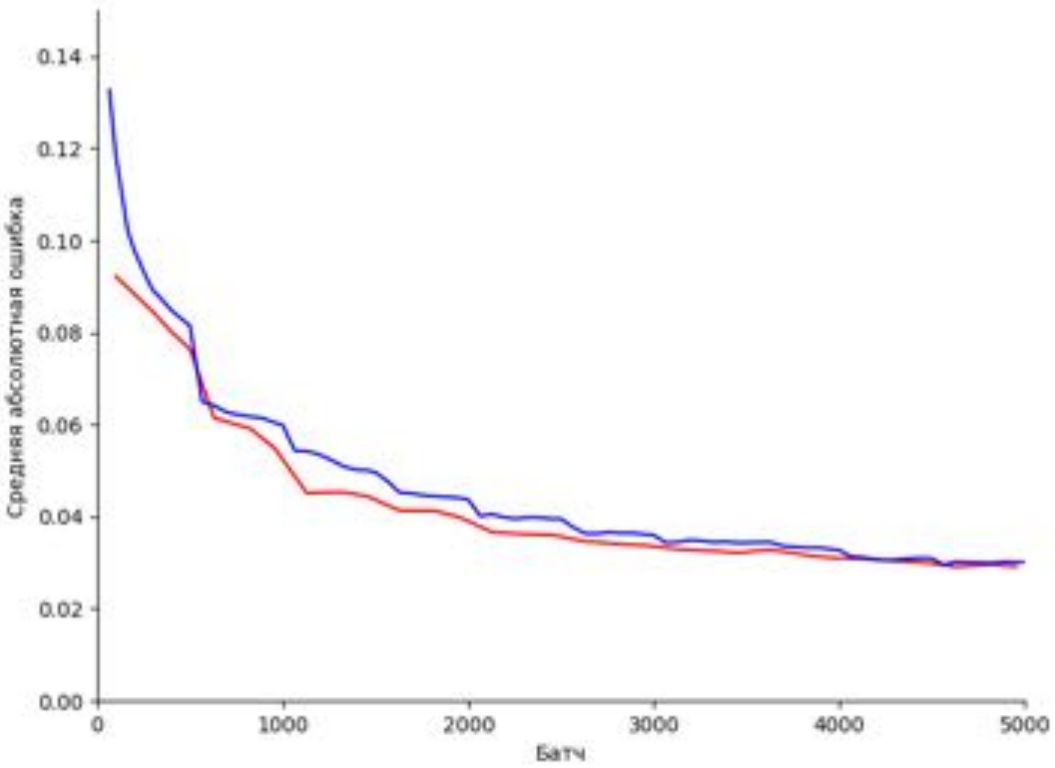


Рисунок 10. Зависимость функции потерь сети-генератора от номера батча.

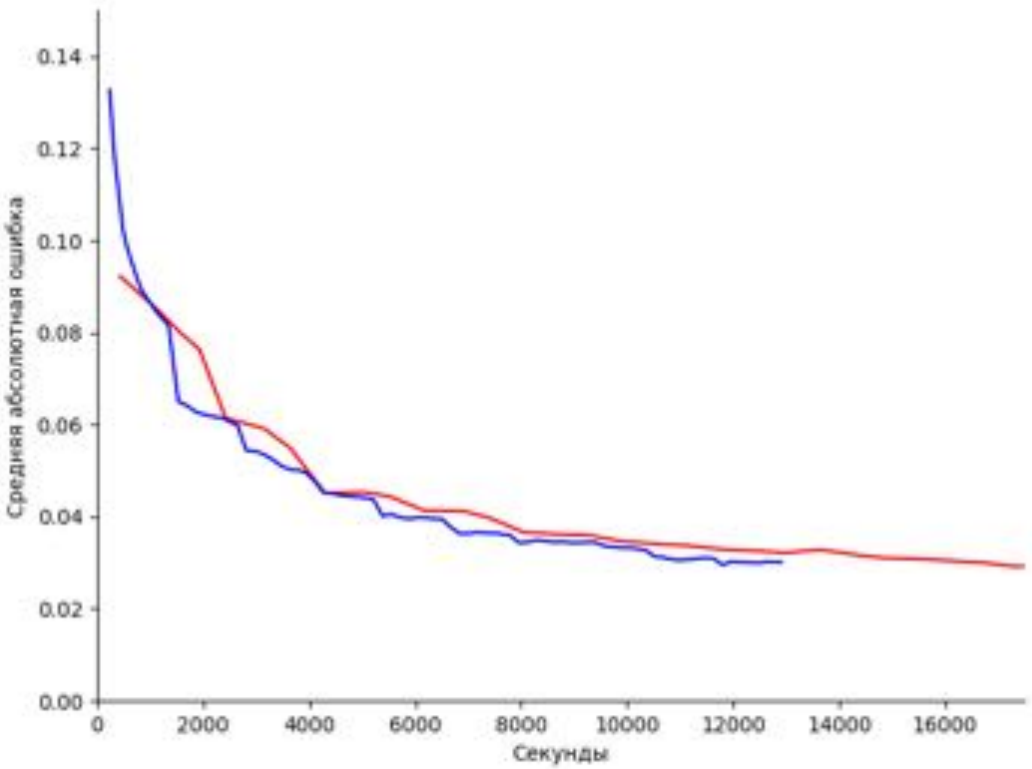


Рисунок 11. Зависимость функции потерь сети-генератора от времени.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы были получены следующие результаты:

- Изучены современные подходы в области искусственных нейронных сетей, включая их программные реализации.
- Реализован и исследован алгоритм обучения конкурирующих сетей в применении к задаче колоризации изображений.
- Проведена серия вычислительных экспериментов с использованием настоящих изображений озера Байкал. В результате была обучена модель конкурирующих сетей, способная создавать правдоподобную колоризацию черно-белых изображений, оригиналы которых никогда не подавались сети на вход.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Arjovsky, Martin. Wasserstein gan [Text] / Martin Arjovsky, Soumith Chintala, Léon Bottou // arXiv preprint arXiv:1701.07875. — 2017.

2. Bugeau, Aurélie. Patch-based image colorization [Text] / Aurélie Bugeau, Vinh-Thong Ta // Pattern Recognition (ICPR), 2012 21st International Conference on / IEEE. — [S. l. : s. n.], 2012. — P. 3058–3061.

3. Charpiat, Guillaume. Automatic image colorization via multimodal predictions [Text] / Guillaume Charpiat, Matthias Hofmann, Bernhard Schölkopf // Computer Vision–ECCV 2008. — 2008. — P. 126–139.

4. Chollet, François. Keras [Text]. — `https://github.com/fchollet/keras`. — 2015.

5. Dropout: A simple way to prevent neural networks from overfitting [Text] / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky [et al.] // The Journal of Machine Learning Research. — 2014. — Vol. 15, no. 1. — P. 1929–1958.

6. Gatys, Leon A. Image style transfer using convolutional neural networks [Text] / Leon A Gatys, Alexander S Ecker, Matthias Bethge // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. — [S. l. : s. n.], 2016. — P. 2414–2423.

7. Generative adversarial nets [Text] / Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza [et al.] // Advances in neural information processing systems. — [S. l. : s. n.], 2014. — P. 2672–2680.

8. Goodfellow, Ian. NIPS 2016 Tutorial: Generative Adversarial Networks [Text] / Ian Goodfellow // arXiv preprint arXiv:1701.00160. — 2016.

9. Goodfellow, Ian. Deep Learning [Text] / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — [S. l.] : MIT Press, 2016. — http://www.deeplearningbook.org.

10. Gybenko, G. Approximation by superposition of sigmoidal functions [Text] / G Gybenko // Mathematics of Control, Signals and Systems. — 1989. — Vol. 2, no. 4. — P. 303–314.

11. Iizuka, Satoshi. Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification [Text] / Satoshi Iizuka, Edgar Simo-Serra, Hiroshi Ishikawa // ACM Transactions on Graphics (TOG). — 2016. — Vol. 35, no. 4. — P. 110.

12. Image-to-image translation with conditional adversarial networks [Text] / Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A Efros // arXiv preprint arXiv:1611.07004. — 2016.

13. Improved Techniques for Training GANs [Text] / T. Salimans, I. Goodfellow, W. Zaremba [et al.] // arXiv preprint arXiv:1606.03498. — 2016.

14. Improved Training of Wasserstein GANs [Text] / Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky [et al.] // arXiv preprint arXiv:1704.00028. — 2017.

15. Ioffe, S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Text] / S. Ioffe, C. Szegedy // arXiv preprint arXiv:1502.03167. — 2017.

16. Kingma, Diederik. Adam: A method for stochastic optimization [Text] / Diederik Kingma, Jimmy Ba // arXiv preprint arXiv:1412.6980. — 2014.

17. Krizhevsky, Alex. Imagenet classification with deep convolutional neural networks [Text] / Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton // Advances in neural information processing systems. — [S. l. : s. n.], 2012. — P. 1097–1105.

18. Levin, Anat. Colorization using optimization [Text] / Anat Levin, Dani Lischinski, Yair Weiss // ACM Transactions on Graphics (ToG) / ACM. — Vol. 23. — [S. l. : s. n.], 2004. — P. 689–694.

19. Papert, Seymour A. The summer vision project [Text] / Seymour A Papert. — 1966.

20. Robertson, Alan R. The CIE 1976 Color-Difference Formulae [Text] / Alan R Robertson // Color Research & Application. — 1977. — Vol. 2, no. 1. — P. 7–11.

21. Ronneberger, Olaf. U-net: Convolutional networks for biomedical image segmentation [Text] / Olaf Ronneberger, Philipp Fischer, Thomas Brox // International Conference on Medical Image Computing and Computer-Assisted Intervention / Springer. — [S. l. : s. n.], 2015. — P. 234–241.

22. Sapiro, Guillermo. Inpainting the colors [Text] / Guillermo Sapiro // Image Processing, 2005. ICIP 2005. IEEE International Conference on / IEEE. — Vol. 2. — [S. l. : s. n.], 2005. — P. II–698.

23. Speeded-up robust features (SURF) [Text] / Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool // Computer vision and image understanding. — 2008. — Vol. 110, no. 3. — P. 346–359.

24. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative

Adversarial Networks [Text] / Han Zhang, Tao Xu, Hongsheng Li [et al.] // arXiv preprint arXiv:1612.03242. — 2016.

25. Tieleman, Tijmen. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude [Text] / Tijmen Tieleman, Geoffrey Hinton // COURSERA: Neural networks for machine learning. — 2012. — Vol. 4, no. 2.

26. Zhang, Richard. Colorful image colorization [Text] / Richard Zhang, Phillip Isola, Alexei A Efros // European Conference on Computer Vision / Springer. — [S. l. : s. n.], 2016. — P. 649–666.

ПРИЛОЖЕНИЕ 1.