

*Федеральное государственное автономное образовательное учреждение высшего  
образования «Московский физико-технический институт (национальный  
исследовательский университет)»*

## Проект по ООП

Выполнили:

Белоусов Анатолий  
Дашевский Дмитрий  
Ильина Анастасия

ФБМФ

Москва, Зеленоград, Екатеринбург, 2020 г.

# Содержание

<b>1 Abstract</b>	<b>3</b>
<b>2 Введение</b>	<b>3</b>
<b>3 Известные результаты</b>	<b>3</b>
3.1 Ограничения традиционных компьютерных программ . . . . .	3
<b>4 Методы</b>	<b>4</b>
4.1 Общая теория . . . . .	4
4.1.1 Машинное обучение . . . . .	4
4.1.2 Нейрон . . . . .	5
4.1.3 Нейросети с прямым распространением сигнала . . . . .	5
4.1.4 Нейроны с сигмоидой, гиперболическим тангенсом и усеченные линейные . . . . .	5
4.2 Полносвязная нейронная сеть . . . . .	6
4.2.1 Градиентный спуск . . . . .	6
4.2.2 Темп обучения . . . . .	7
4.2.3 Проблема переобучения . . . . .	7
4.2.4 Борьба с переобучением в глубоких нейросетях . . . . .	8
4.3 Сверточная нейронная сеть . . . . .	9
4.3.1 Недостатки выбора признаков . . . . .	9
4.3.2 Фильтры и карты признаков . . . . .	9
4.3.3 Полное описание сверточного слоя . . . . .	10
4.3.4 Max Pooling . . . . .	11
4.3.5 U-Net . . . . .	11
4.4 Предобученные нейросети . . . . .	12
4.4.1 AlexNet . . . . .	12
4.4.2 ResNet . . . . .	12
4.4.3 VGG Net . . . . .	13
4.4.4 DenseNet . . . . .	13
<b>5 Реализации</b>	<b>14</b>
5.1 Реализация полносвязной нейронной сети . . . . .	14
5.1.1 Изменим количество слоев и эпох . . . . .	14
5.1.2 Поменяем функцию активации и соседние гиперпараметры . . . . .	16
5.1.3 Изменим batch size . . . . .	17
5.1.4 Изменим процент тестовой выборки . . . . .	18
5.2 Реализация U-Net . . . . .	18
5.2.1 Рассмотрим первую, самую простую, выборку . . . . .	19
5.2.2 Посмотрим другие выборки . . . . .	20
5.2.3 Поменяем оптимизаторы программы . . . . .	20
5.2.4 Поменяем loss function программы . . . . .	20
5.3 Реализация предобученных нейросетей . . . . .	22
<b>6 Обсуждение</b>	<b>26</b>

# 1. Abstract

В данном проекте мы создали нейронную сеть, которая способна распознавать изображения клеток и выделять их на фоне остальной информации на изображении. Также мы создали несколько сетей, задачей которых было по изображению определять непосредственное число клеток на изображении. Во время выполнения мы построили несколько вариаций таких сетей: полносвязанную и сверточную, для того, чтобы понять, какая модель больше подходит для наших исследований. Автоматизация определения количества клеток (а в перспективе и отрисовка их границ) позволит быстро анализировать большой массив данных для внесение в базу или для дальнейших исследований. В этой статье мы много времени уделили поискам оптимальных параметров для получения более благоприятных результатов.

## 2. Введение

Сегодня многие биологические исследования подразумевают обработку больших данных, в частности, работу с изображениями клеток. Чтобы упростить получение результатов, можно написать нейронную сеть, которая будет делать монотонную работу за исследователя, автоматически выводя результаты с большой точностью. Нейросети – очень перспективное направление, которое позволяет выполнять практически любые задачи. Работа с изображениями как раз одна из таких задач. Сейчас существует ряд программ выполняющих схожие функции, однако их недостаток заключается в ориентированности на определённые форматы изображений, привязанность к аппаратному способу получения этих данных (программы чаще пишутся производителями оборудования или совместно с ними), а также в обработке только однотипных изображений. Нашей целью являлось создание сети применимой в лабораторной практике без использования дорогостоящего программного обеспечения и изображений повышенного качества, а также нацеленной на быструю и массовую обработку данных.

Мы прошли путь от простейшей полносвязной нейронной сети, разобрали все основные понятия и нюансы ее построения, перейдя к сверточным нейросетям, которые являются более сложными, но более приспособленными для работы с изображениями. Разобравшись с теоретической подоплекой поставленной задачи, мы перешли к реализации, чтобы показать, что наши идеи воплотимы в жизнь.

## 3. Известные результаты

Для обработки изображений клеток специалисты пользуются приложением ImageJ или Fiji. Эти программы неудобны тем, что необходимо обрабатывать картинки по одной, на что тратится огромное количество времени. Весь функционал данных инструментов можно реализовать на языке Python3.

### 3.1. Ограничения традиционных компьютерных программ

Стандартные программы доказали свою состоятельность в двух областях: они очень быстро ведут вычисления, они неукоснительно следуют инструкциям.

Представим, что нам надо сделать что-то интереснее математических расчетов – написать программу для автоматического распознавания почерка. Возьмем за основу рисунок (1).

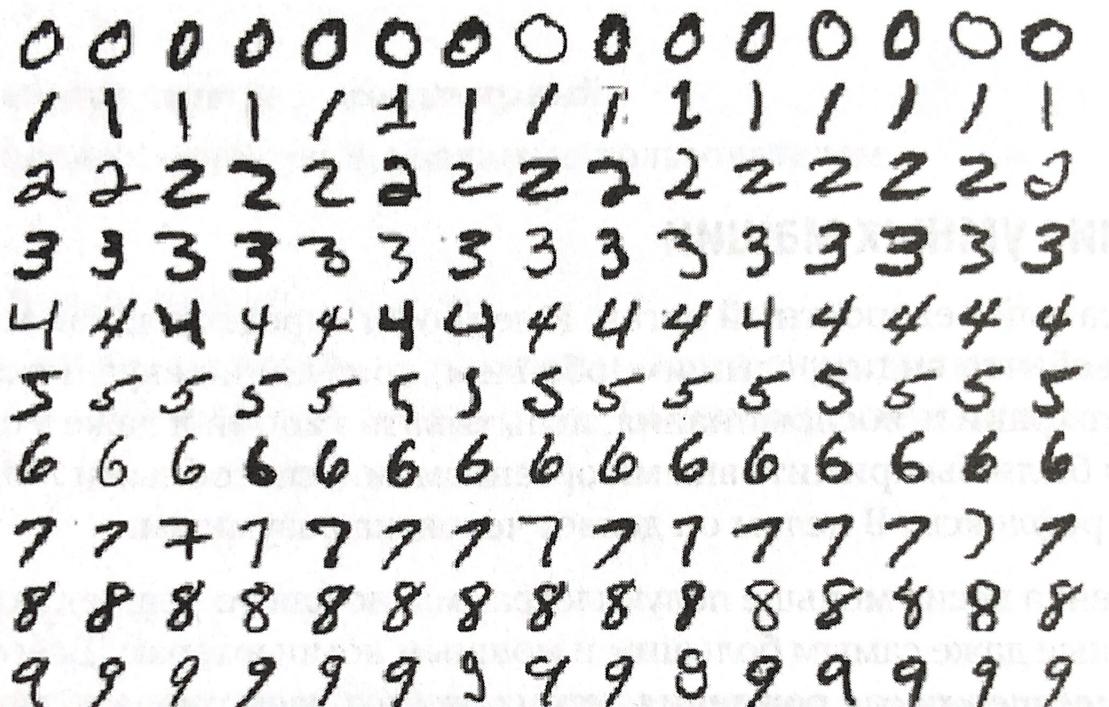


Рисунок 1 – Изображения из массива рукописных данных MNIST

Хотя каждая цифра на рисунке отличается от предыдущей, мы легко опознаем в первом ряде нули, во втором единицы и тд. Теперь напишем компьютерную программу, которая решит ту же задачу. Какие правила нужно задать, чтобы различать цифры?

Начнем с простого. Например, укажем, что нулю соответствует изображение окружного замкнутого контура. Все примеры с рисунка (1), кажется, удовлетворяют этому определению, но таких признаков недостаточно. Что, если у кого-то ноль – не всегда замкнутая фигура? И как отличить такой ноль от шестерки?

Можно задать рамки расстояния между началом и концом петли, но не очень понятно какие. И это только начало проблем. Как различить тройки и пятерки? Четверки и девятки? Можно добавлять правила или признаки, после тщательных наблюдений и месяцев проб и ошибок, но понятно одно: процесс будет нелегок.

## 4. Методы

Чтобы получить наиболее многогранные и исчерпывающие результаты, мы решили использовать как модель полносвязной НС, так и модель сверточной НС.

### 4.1. Общая теория

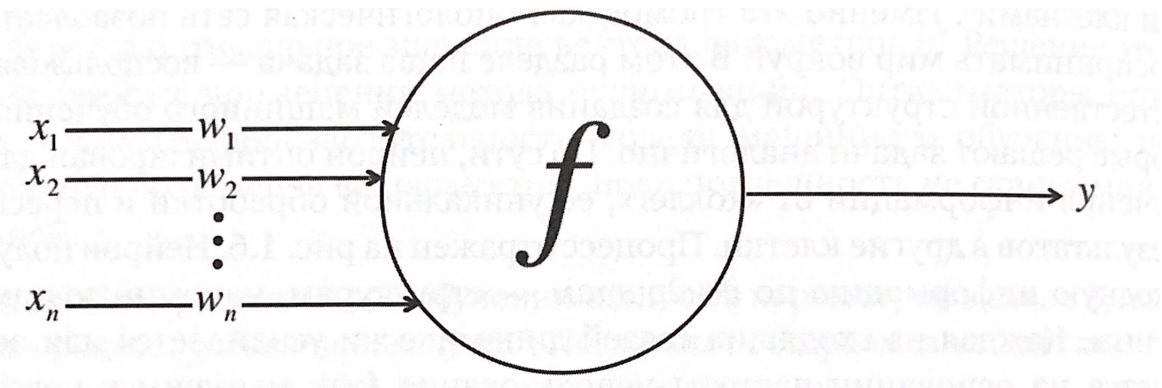
#### 4.1.1. Машинное обучение

Для решения задач, как например, определение рукописных цифр, нужен совсем другой подход. Глубокое обучение – отрасль более широкой области исследования искусственного интеллекта: *машинного обучения*, подразумевающего получение знаний из примеров. Мы не задаем компьютеру огромный список правил решения задачи, а предоставляем модель, с помощью которой он может сравнивать примеры, и краткий набор инструкций для ее модификации в случае ошибки. [1]

#### 4.1.2. Нейрон

Нейрон – основная единица мозга. Небольшой его фрагмент, размером примерно с рисовую зернышко, содержит более 10000 нейронов, каждый из которых в среднем формирует 6000 связей с другими такими клетками. [2] Именно эта громоздкая биологическая сеть позволяет нам воспринимать мир вокруг. По сути, нейрон оптимизирован для получения информации от «коллег», ее уникальной обработки и пересылки в другие клетки. Нейрон получает входную информацию по дендритам. Каждая из входящих связей динамически усиливается или ослабляется на основании частоты использования и сила соединений определяет вклад входящего элемента информации в то, что нейрон выдаст на выходе.

Мы можем преобразовать функциональное понимание работы нейронов в нашем мозге в искусственную модель на компьютере. Как и биологические нейроны, искусственный получает некоторый объем входных данных –  $x_1, x_2, \dots, x_n$ , каждый элемент которых умножается на определенное значение веса –  $w_1, w_2, \dots, w_n$ . Эти значения, как и раньше, суммируются, давая *логит* нейрона:  $z = \sum_{i=0}^n w_i x_i$ . Логит проходит через функцию активации  $f$ , образуя выходное значение  $y = f(z)$ . [3]



**Рисунок 2** – Схема работы нейрона в искусственной нейросети

Представляем входные данные нейрона как вектор  $\mathbf{x} = [x_1, x_2, \dots, x_n]$ , а веса нейрона как  $\mathbf{w} = [w_1, w_2, \dots, w_n]$ . Теперь выходные данные нейрона можно выразить как  $y = f(\mathbf{x} \cdot \mathbf{w} + b)$ , где  $b$  – смещение.

#### 4.1.3. Нейросети с прямым распространением сигнала

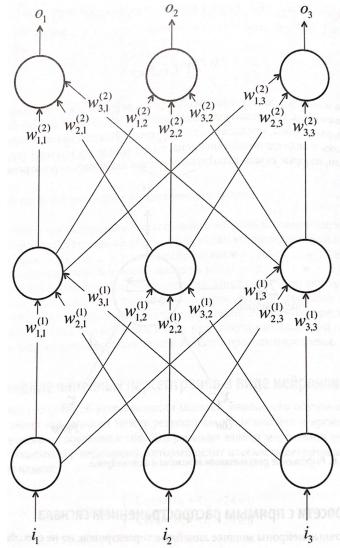
Одиночные нейроны мощнее линейных персепtronов, но не способны решить сложные проблемы обучения.

В нижний слой поступают входные данные. Верхний (выходные узлы) вычисляет ответ. Средний слой нейронов называется скрытым, и здесь  $w_{i,j}^{(k)}$  – вес соединения  $i$ -го нейрона в  $k$ -м слое с  $j$ -м нейроном в  $(k+1)$ -м слое. Эти веса определяют вектор параметров  $\theta$ , и, как и ранее, наша способность решать задачи при помощи нейросетей зависит от нахождения оптимальных значений для  $\theta$ .

#### 4.1.4. Нейроны с сигмоидой, гиперболическим тангенсом и усеченные линейные

На практике для вычислений применяются три типа нелинейных нейронов, но мы рассмотрим только два. Первый называется сигмоидным и использует функцию:

$$f(z) = \frac{1}{1 + e^{-z}}.$$



**Рисунок 3** – Простой пример нейросети с прямым распространением сигнала с тремя слоями и тремя нейронами на каждой слой

Интуитивно это означает, что если логит очень мал, выходные данные логического нейрона близки к 0. Если логит очень велик, то к 1. Между этими двумя экстремумами нейрон принимает форму буквы S.

Нейроны гиперболического тангенса используют похожую S-образную нелинейность, но исходящие значения варьируются не от 0 до 1, а от -1 до 1. Формула для их предсказания  $f(z) = \tanh z$ . Когда используют S-образные нейросети, часто предпочитают tanh-нейроны, а не сигмоидные, поскольку у tanh-нейронов центр находится в 0.

Еще один тип нелинейности используется нейроном с *усеченным линейным преобразованием* (ReLU). Здесь задействована функция  $f(z) = \max(0, z)$ , и ее график имеет форму хоккейной клюшки.

ReLU в последнее время часто выбирается для выполнения многих задач (особенно в системах компьютерного зрения) по ряду причин, несмотря на свои недостатки. [4]

Мы хотим обучить нейрон и подбираем оптимальные веса, чтобы свести к минимуму ошибки при распознавании примеров. Можно сказать, мы хотим свести к минимуму квадратичную ошибку во всех примерах, которые встретим. Формально, если мы знаем, что  $t^{(i)}$  – верный ответ на  $i$ -й пример, а  $y^{(i)}$  – значение, вычисленное нейросетью, мы хотим свести к минимуму значение функции потерь  $E$ :

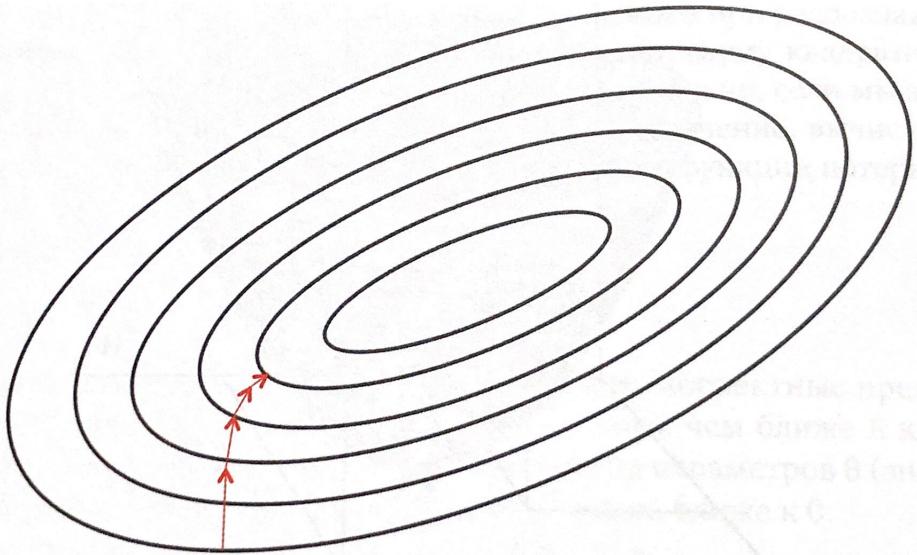
$$E = \frac{1}{2} \sum_i (t^{(i)} - y^{(i)})^2.$$

Квадратичная модель равна 0, когда модель дает корректные предсказания для каждого обучающего примера. Более того, чем ближе  $E$  к 0, тем лучше модель. Наша цель – выбрать такой вектор параметров  $\theta$ , чтобы  $E$  было как можно ближе к 0.

## 4.2. Полносвязная нейронная сеть

### 4.2.1. Градиентный спуск

Допустим, у линейного нейрона есть только два входа. Мы можем представить себе трехмерное пространство, в котором горизонтальные измерения соответствуют  $w_1$  и  $w_2$ , а вертикальное – значению функции потерь  $E$ . Таким образом мы получим параболоид



**Рисунок 4 – Визуализация поверхности ошибок как набора контуров**

вращения. Будем работать с двумерным пространством, где измерения соответствуют весам. Контуры сопоставлены значениям  $w_1$  и  $w_2$ , которые дают одно и то же  $E$ . Чем ближе они друг к другу, тем круче уклон. Направление самого крутого уклона всегда перпендикулярно контурам. Его можно выразить в виде вектора, называемого *градиентом*.

Допустим, мы случайным образом инициализировали веса сети, оказавшись где-то на горизонтальной поверхности. Оценив градиент в текущей позиции, мы можем найти направление самого крутого спуска и сделать шаг в нем. Теперь мы на новой позиции, которая ближе к минимуму, чем предыдущая. Мы производим переоценку направления самого крутого спуска, взяв градиент, и делаем шаг в новом направлении. Этот алгоритм известен как градиентный спуск, и мы будем использовать его для решения проблемы обучения. [5]

#### 4.2.2. Темп обучения

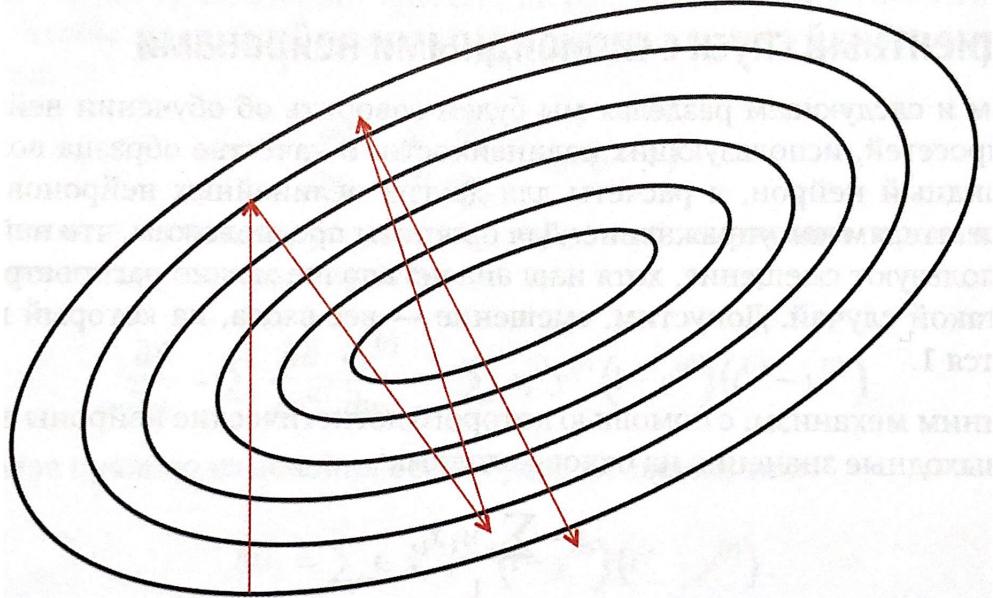
Обратим внимание на гиперпараметры. Помимо весов, определенных в нашей нейросети, обучающим алгоритмам нужен ряд дополнительных параметров. Один из этих гиперпараметров – темп обучения.

На каждом шаге движения перпендикулярно контуру нам нужно решать, как далеко мы хотим зайти, прежде чем заново вычислять направление. Это расстояние зависит от крутизны поверхности. Чем ближе мы к минимуму, тем меньше должен быть шаг. Мы понимаем, что мы близки к минимуму, поскольку поверхность намного более плоская и крутизну мы используем как индикатор степени близости к этому минимуму. Но если поверхность ошибки рыхлая, процесс может занять много времени. Поэтому часто стоит умножить градиент на масштабирующий коэффициент – темп обучения. Его выбор – сложная задача.

Если он будет слишком мал, возможно, процесс займет слишком много времени. Но если темп будет слишком высоким, то кончится это, скорее всего, тем, что мы отклонимся от минимума.

#### 4.2.3. Проблема переобучения

У нас есть ряд точек на плоской поверхности, задача – найти кривую, которая наилучшим образом опишет набор данных. Используя эти данные, мы обучаем две модели:



**Рисунок 5** – Если темп обучения слишком велик, возникают проблемы со сходимостью

линейную и многочлен 12-й степени. Какой кривой стоит доверять?

Линейная модель не только субъективно, но и количественно лучше (по показателю квадратичной ошибки). Но это ведет к очень интересному выводу по поводу усвоения информации и оценки моделей машинного обучения. Стряя очень сложную модель, легко полностью подогнать ее к обучающему набору данных. Ведь мы ей даем достаточно степеней свободы для искажения, чтобы вписаться в имеющиеся значения. Но когда мы оцениваем такую модель на новых данных, она работает плохо, то есть, плохо обобщает. Это явление называется *переобучением*.

#### 4.2.4. Борьба с переобучением в глубоких нейросетях

Есть несколько методов борьбы с переобучением. Один из них носит название *регуляризации*. Он изменяет целевую функцию, которую мы минимализируем, добавляя условия, которые препятствуют появлению больших весов. Иными словами, мы изменяем целевую функцию на  $Error + \lambda F(\theta)$ , где  $f(\theta)$  увеличивается, когда компоненты  $\theta$  растут, а  $\lambda$  – показатель регуляризации. Значение  $\lambda$  определяет, в какой степени мы хотим защититься от переобучения. Если  $\lambda = 0$ , мы не принимаем никаких мер. Если  $\lambda$  слишком велико, приоритетом модели будет сохранение  $\theta$  на низком уровне, а не нахождение значений параметров, которые дадут хорошие результаты на обучающем наборе.

Самый распространенный тип регуляризации в машинном обучении – так называемая *L2*-регуляризация. [6] Ее можно провести, дополнив функцию потерь квадратом величины всех весов в нейросети. Иными словами, для каждого веса  $w$  в нейросети мы добавляем  $\frac{1}{2}\lambda w^2$  в функцию потерь. *L2*-регуляризация интуитивно интерпретируется как препятствующая появлению пиковых векторов весов и предпочитающая равномерные векторы весов.

К тому же в ходе градиентного спуска использование *L2*-регуляризации в целом означает, что каждый вес линейно уменьшается до 0. Благодаря этому феномену *L2*-регуляризации получила второе название: *сокращение весов*.

Еще один распространенный вариант – *L1*-регуляризации. Здесь мы добавляем значение  $\lambda|w|$  для каждого веса  $w$  в нейросети. *L1*-регуляризации обладает интригующим свойством: в ходе оптимизации векторы весов становятся очень разреженными. Иными словами, нейроны начинают использовать небольшое количество самых важных входов и становятся устойчивыми к шуму на входе. А векторы весов, полученные *L2*-регуляризации,

обычно равномерны и невелики.  $L1$ -регуляризация очень полезна, когда вы хотите понять, какие именно свойства вносят вклад в принятие решения. Если такой уровень анализа свойств не нужен, мы используем  $L2$ -регуляризацию: она на практике работает лучше.

Совсем иной метод борьбы с переобучением – *прореживание* (*Dropout*), который особенно популярен у специалистов по глубоким нейросетям. [7] При обучении он используется так: нейрон становится активным только с некой вероятностью  $p$ , иначе его значение приравнивается к 0. На интуитивном уровне можно решить, что это заставляет нейросеть оставаться точной даже в условиях недостатка информации. Сеть перестает быть слишком зависимой от отдельного нейрона или их небольшого сочетания. С точки зрения математики прореживание препятствует переобучению, давая возможность приблизительно сочетать экспоненциально большое количество архитектур нейросетей, причем эффективно.

## 4.3. Сверточная нейронная сеть

Человеческое зрение основано на нейронах. Последние отвечают за доставку световой информации в глаза. [8] Эта информация проходит здесь предварительную обработку, препровождается в зрительную кору мозга и только там анализируется. За все эти функции отвечают нейроны. Поэтому интуитивно кажется, что стоило бы обобщить модели нейронных сетей для создания более эффективных систем компьютерного зрения.

### 4.3.1. Недостатки выбора признаков

Начнем с простой проблемы компьютерного зрения. Нам надо определить, есть ли на изображении человеческое лицо. Именно эту проблему затрагивали Пол Виола и Майкл Джонс в своей основополагающей работе в 2001 году. [9]

Люди долго пытались решить эту проблему с помощью классической техники машинного обучения – через распознавание классов. Но алгоритм на самом деле так и не научился «смотреть». Помимо различий в интенсивности света, наш мозг использует обширный спектр визуальных подсказок для определения того, что мы видим человеческое лицо.

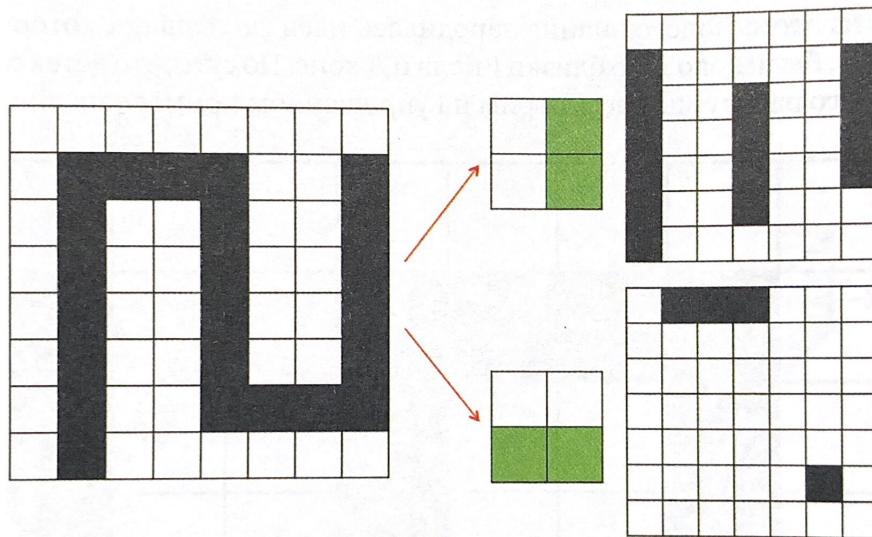
В 2012 году Алекс Крижевский из лаборатории Джеффри Хинтона в Университете Торонто впервые применил архитектуру глубокого обучения, ныне известную как *сверточная нейросеть*, к проблемам особого масштаба и сложности.

### 4.3.2. Фильтры и карты признаков

Первой зародилась идея *фильтра*, к которой, как оказалось, были довольны близки Виола и Джонс. По сути, это детектор признака.

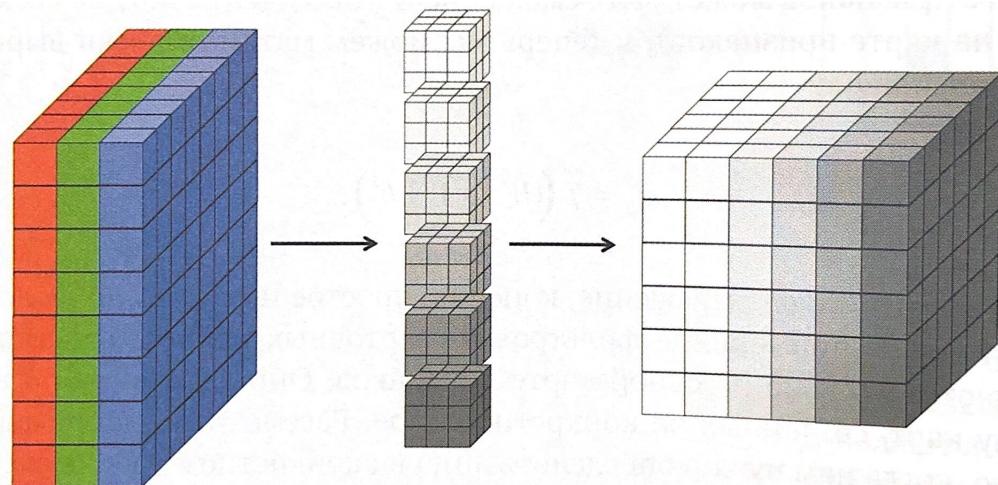
Допустим, нам нужно определить вертикальные и горизонтальные линии на этом изображении. Один из вариантов решения – использовать подходящий детектор признаков. Например, чтобы определить вертикальные линии, воспользуемся детектором сверху, распространив его на все изображение, и на каждом шаге будем проверять совпадения. Запишем ответы в матрицу в правом верхнем углу. Если есть совпадение, мы красим соответствующую ячейку в черный цвет, если нет – оставляем белой. В результате получим карту признаков, которая показывает, где мы нашли нужный признак в исходном изображении. То же можно сделать с детектором горизонтальных линий снизу.

Это называется сверткой. Мы берем фильтр и распространяем его на всю область входящего изображения. Обозначим  $k$ -ю карту признаков в слое как  $t$  как  $t^k$ . Далее обозначим соответствующий фильтр по значениям его весов  $W$ . Предложим, что у нейронов на карте признаков может быть смещение  $b^k$ , и теперь мы можем математически выразить карту признаков:



**Рисунок 6 – Применение фильтров для вертикальных и горизонтальных линий**

$$m_{ij}^k = f((W \cdot x)_{ij} + b^k).$$



**Рисунок 7 – Трехмерная визуализация сверточного слоя, где каждый фильтр соответствует сектору в получившемся выходном объему**

#### 4.3.3. Полное описание сверточного слоя

Во-первых, он принимает следующие входящие характеристики:

- Ширина  $w_{ij}$ .
- Высота  $h_{ij}$ .
- Глубина  $d_{ij}$ .
- Дополнение нулями  $p$ .

Входные значения обрабатываются фильтрами в количестве  $k$ , которые соответствуют весам и связям в сверточной сети. У фильтров есть ряд гиперпараметров, которые описываются так:

- Пространственная протяженность  $e$ , равная высоте и ширине фильтра.
- Сдвиг  $S$ , или дистанция между последовательными применениями фильтра к объему входных данных. Если использовать сдвиг 1, получится полная свертка, описанная в предыдущем разделе.
- Смещение  $b$

Получаем выходные данные со следующими характеристиками:

- Функция активации  $f$ , применяемая ко входному логиту каждого нейрона в объеме входных данных для определения выходного значения.
- Ширина  $w_{out} = \left\lceil \frac{w_{in}-e+2p}{s} \right\rceil + 1$ .
- Высота  $h_{out} = \left\lceil \frac{h_{in}-e+2p}{s} \right\rceil + 1$ .
- Глубина  $d_{out} = k$ .

#### 4.3.4. Max Pooling

Чтобы резко уменьшить размерность карт признаков и заострить внимание на найденных признаках, мы иногда используем слой *max pooling* после сверточного. Смысл его – в разбиении карты на сектора равного размера. Так мы получаем сжатую карту признаков. Фактически мы создаем свою ячейку для каждого сектора, вычисляем максимальное значение по нему и записываем его в соответствующие ячейку сжатой карты признаков.

Интересное свойство *max pooling* – *локальная инвариантность*. Даже если входные значения немного варьируются, выходные остаются неизменными.

#### 4.3.5. U-Net

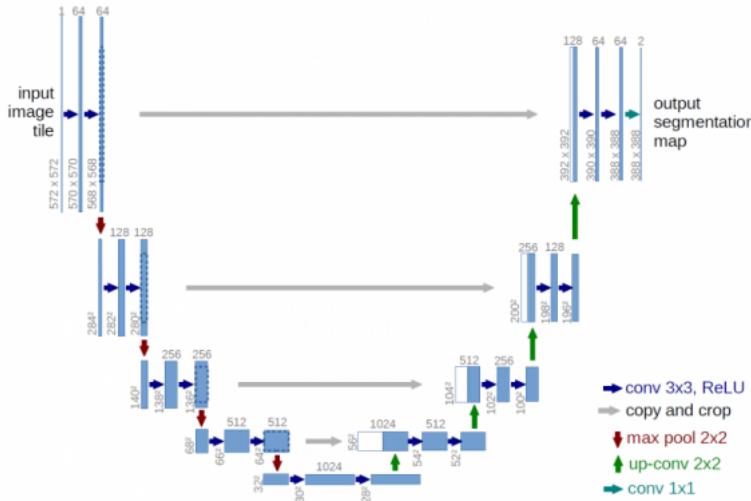
Мы решили использовать сверточную НС U-net. U-Net считается одной из стандартных архитектур CNN для задач сегментации изображений, когда нужно не только определить класс изображения целиком, но и сегментировать его области по классу, т.е. создать маску, которая будет разделять изображение на несколько классов. Архитектура состоит из стягивающего пути для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию.

Архитектура сети представлена на данной схеме:

На каждом этапе поникающей дискретизации каналы свойств удваиваются. Каждый шаг в расширяющемся пути состоит из операции повышающей дискретизации карты свойств, за которой следуют:

- Свертка  $2 \times 2$ , которая уменьшает количество каналов свойств;
- объединение с соответствующим образом обрезанной картой свойств из стягивающегося пути;
- две  $3 \times 3$  свертки, за которыми следует ReLU.

Сеть обучается методом стохастического градиентного спуска на основе входных изображений и соответствующих им карт сегментации. Из-за сверток выходное изображение меньше входного сигнала на постоянную ширину границы. Применяемая попиксельно, функция soft-max вычисляет энергию по окончательной карте свойств вместе с функцией кросс-энтропии. Кросс-энтропия, вычисляемая в каждой точке, определяется так:



**Рисунок 8 – Архитектура U-Net**

$$E = - \sum_x \omega(x) \log p_{l(x)}(x).$$

Граница разделения вычисляется с использованием морфологических операций. Затем вычисляется карта весовых коэффициентов:

$$\omega(x) = \omega_c(x) + \omega_0 \exp\left(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}\right).$$

#### 4.4. Предобученные нейросети

В настоящее время для уменьшения времени обучения и улучшения качества распознавания классов используются предобученные нейросети. Их отличие заключается в том, что сеть использует веса, полученные при обучении на больших выборках таких как COCO, MNIST и прочие. Также эти выборки уже зарекомендовали себя как эффективные инструменты для распознавания классов на изображениях в соревнованиях таких как Kaggle. Рассмотрим некоторые из них.

##### 4.4.1. AlexNet

Была одной из первых свёрточных нейросетей, позволяющих продемонстрировать преимущество этого вида сетей перед полносвязными. Также отличительной чертой этой сети стало использование функции активации ReLU, которая заменила сигмоиду и гиперболический тангенс. Характерной особенностью сети является использование dropout, что позволяет представить сеть в виде ансамбля сетей и понизить вероятность переобучения.

##### 4.4.2. ResNet

Найдкой этой сети являлось внедрение технологии Shortcut Connections

Суть метода состоит в том, что часть информации с одного слоя сохраняется и "пробрасывается" через несколько слоёв и добавляется к информации прошёлшей через эти слои. Методика позволяет сохранять структуру изначального изображения, при этом выделяя интересующие нас особенности. В результате точность сети не падает при увеличении числа слоёв.

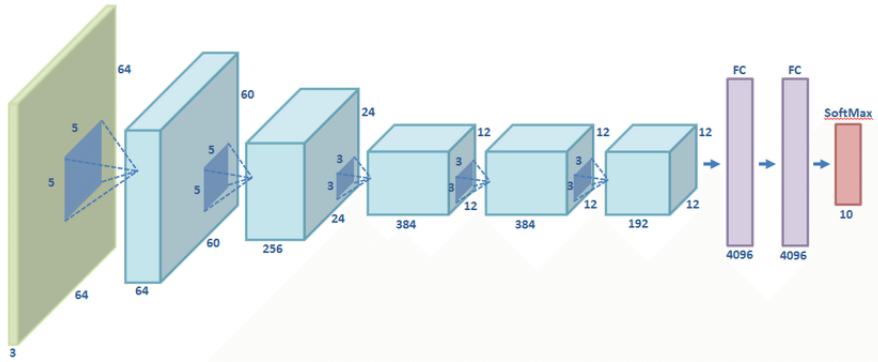


Рисунок 9 – Структура сети AlexNet

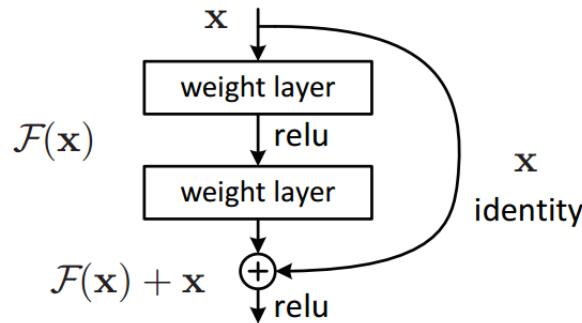


Рисунок 10 – Структура Shortcut Connections

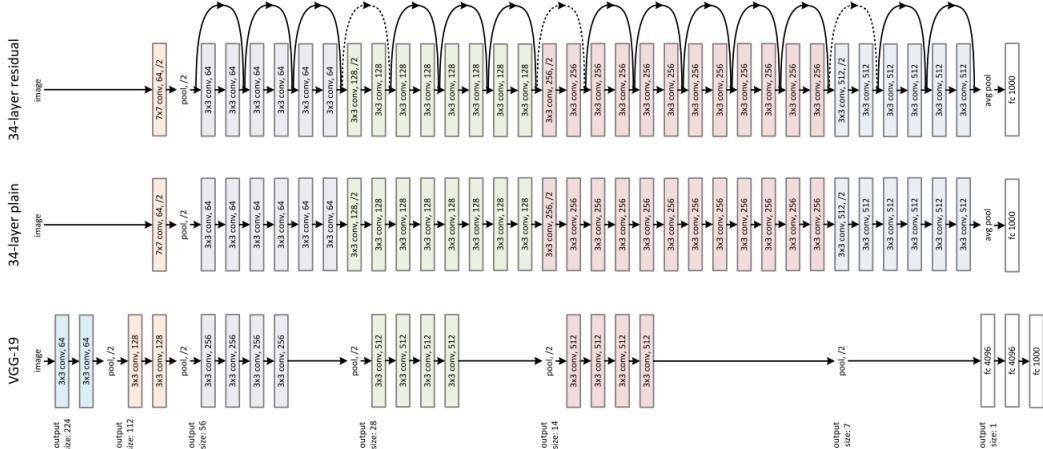


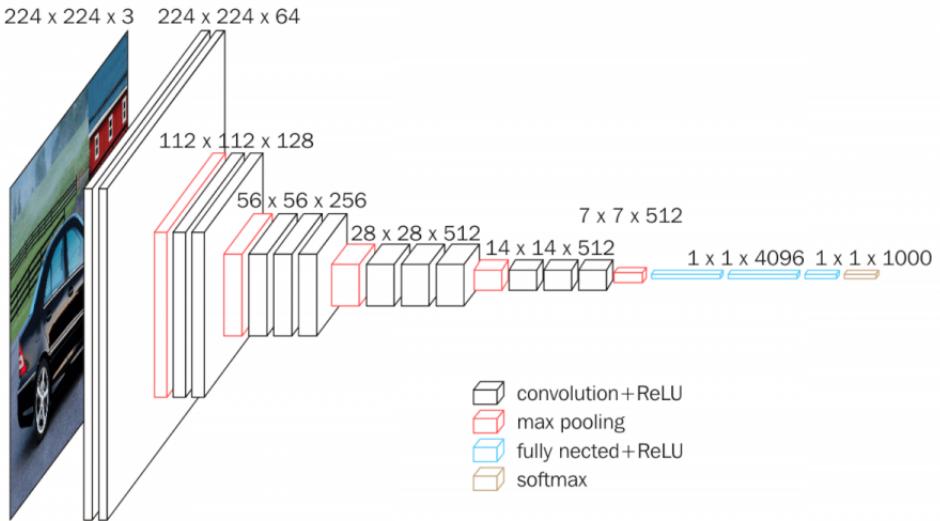
Рисунок 11 – Структура ResNet

#### 4.4.3. VGG Net

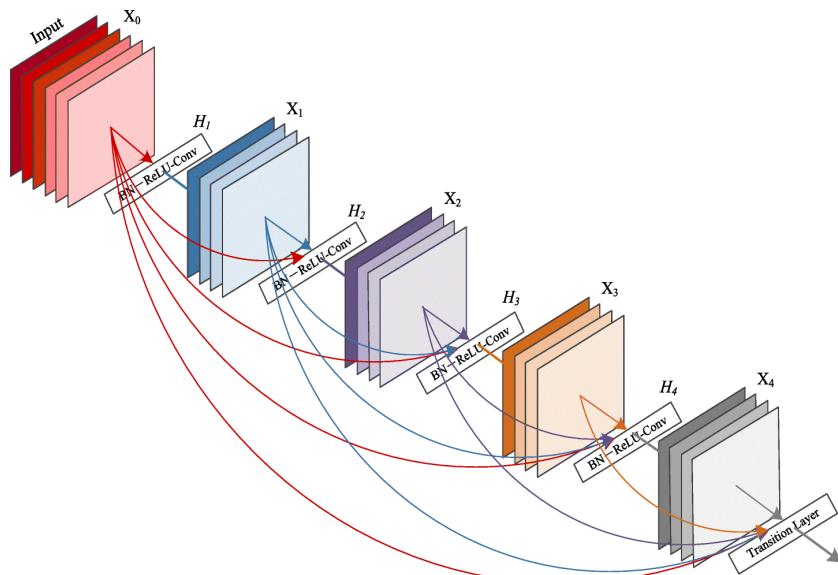
Авторы сети показали, что использование свёрток  $7 \times 7$  эквивалентно использованию трёх последовательных свёрток  $3 \times 3$ , поэтому применение новых свёрток уменьшило количество слоёв и повысило качество предсказаний сети.

#### 4.4.4. DenseNet

Сеть схожа по строению с ResNet, но теперь каждый слой передаёт информацию о себе всем последующим слоям, и сам воспринимает информацию от всех предыдущих слоёв.



**Рисунок 12 – Структура VGG Net**



**Рисунок 13 – Структура DenseNet**

## 5. Реализации

### 5.1. Реализация полносвязной нейронной сети

Изначально наши картинки находятся в разрешении 540 на 412 пикселов, поэтому с помощью библиотеки «Skimage» мы приведем их к квадратному размеру 224 на 224 точки, добавили шумы с 10 различными интенсивностями и повернули каждый картинку 12 раз. Таким образом, в сумме у нас есть 1320 изображений и столько же масок. Теперь мы готовы варьировать параметры.

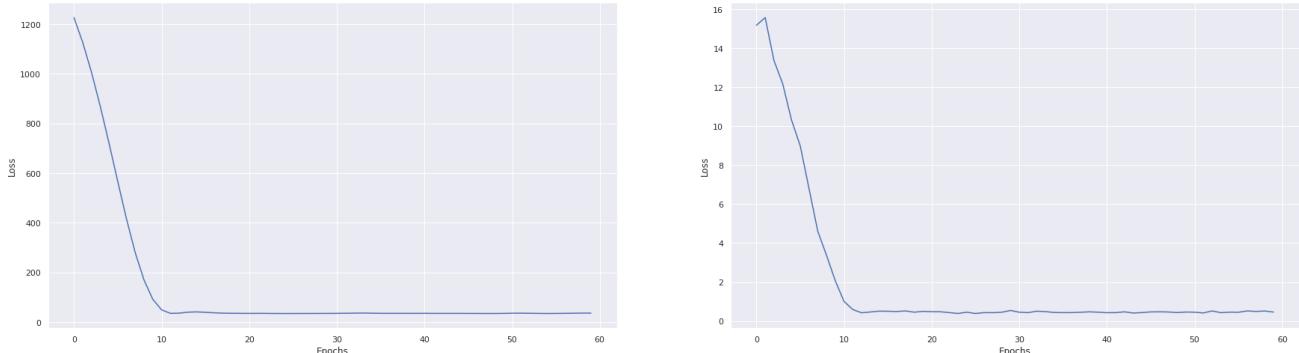
#### 5.1.1. Изменим количество слоев и эпох

За базовую конфигурацию примем нейросеть с двумя слоями, со 100 скрытыми нейронами и функцией активации в виде гиперболического тангенса. Тестовая выборка составляет 30% от всего количества объектов.

При написании всех нейросетей мы использовали библиотеку «PyTorch» как основное

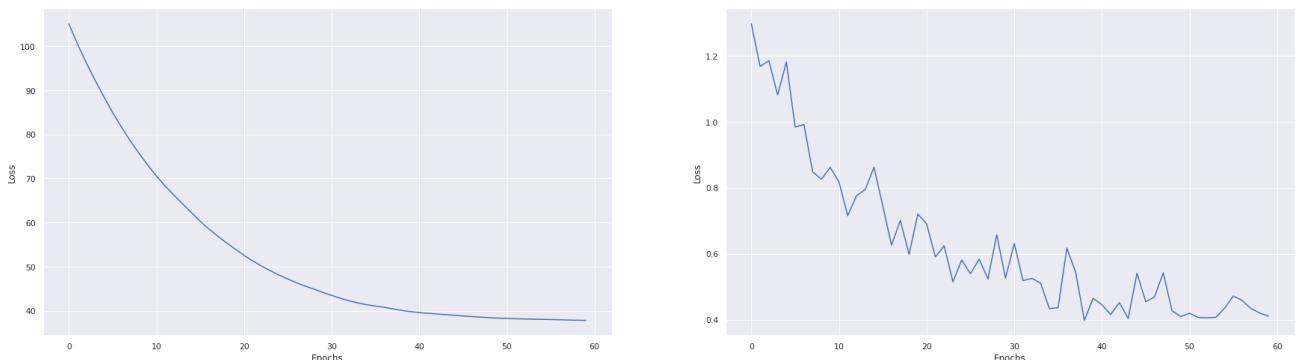
средство. Все операции перекладывались на GPU для ускорения работы и повышения производительности.

Принимая во внимание все вышеописанные условия, получаем такой результат:



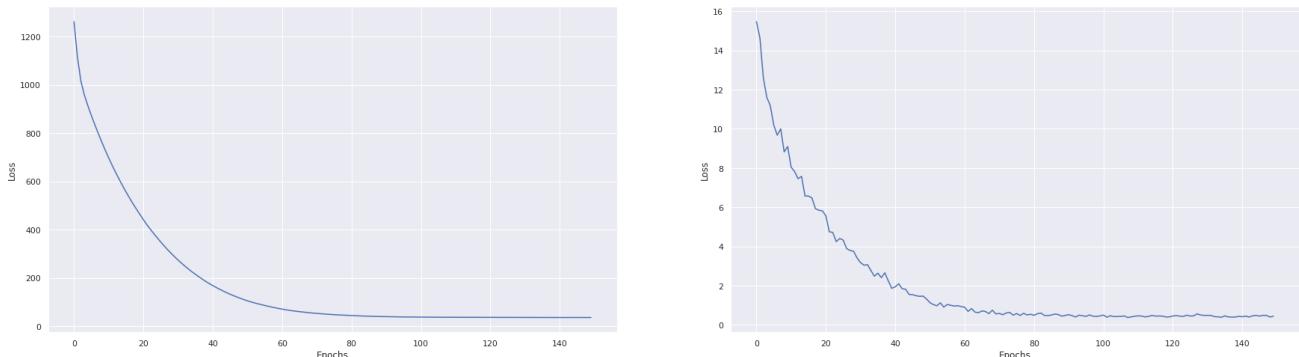
**Рисунок 14** – Базовая конфигурация нашей полносвязной нейронной сети. Справа Test Loss, слева Train Loss

1. Увеличим количество слоев до трех



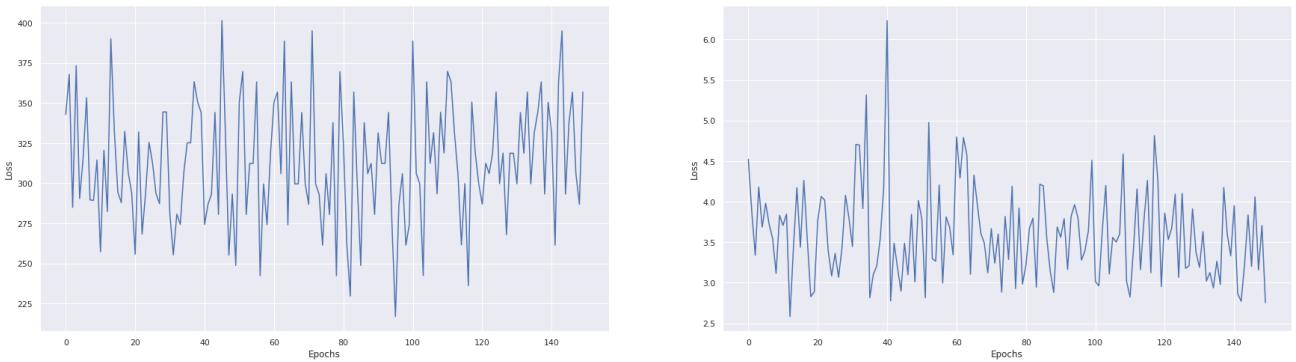
**Рисунок 15** – Три слоя у полносвязной нейросети, 60 эпох. Справа Test Loss, слева Train Loss

2. Увеличим количество эпох до 150.



**Рисунок 16** – Три слоя у полносвязной нейросети, 150 эпох. Справа Test Loss, слева Train Loss

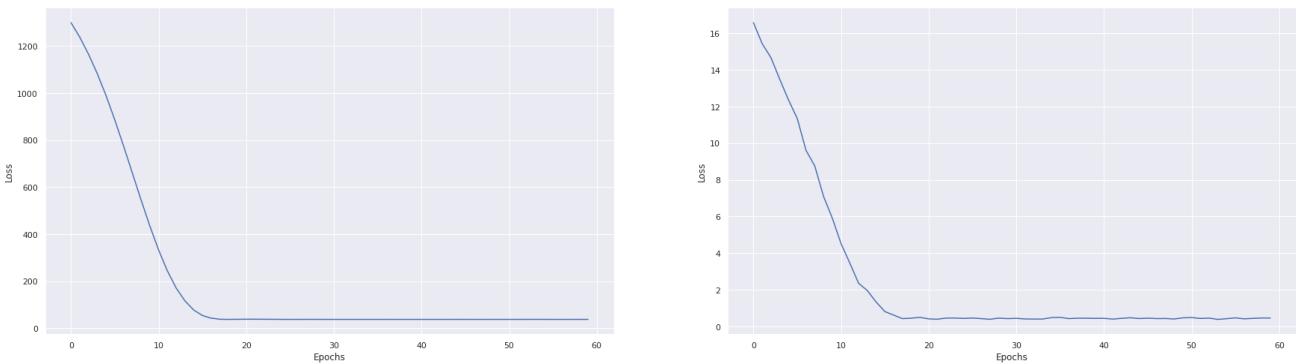
3. Здесь не происходит переобучения, поэтому при применения дропаута результат только испортится, что можно показать.



**Рисунок 17** – Три слоя у полносвязной нейросети, 150 эпох, dropout. Справа Test Loss, слева Train Loss

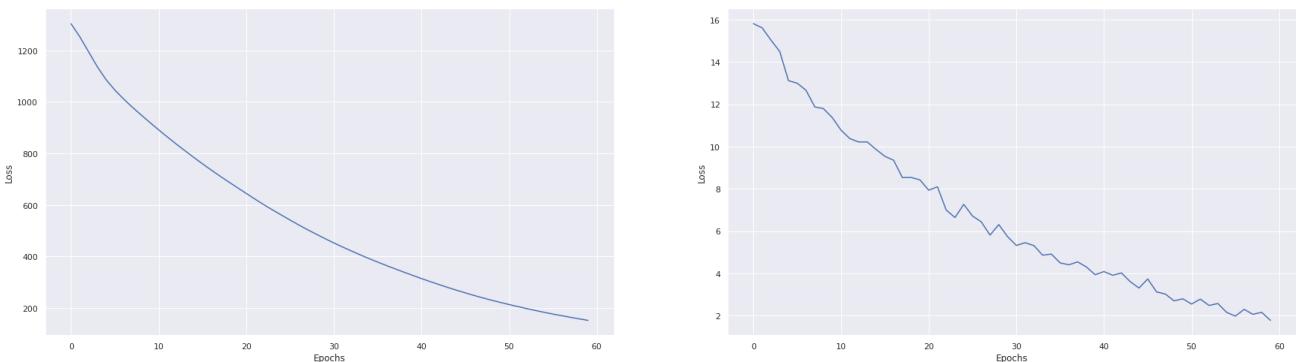
### 5.1.2. Поменяем функцию активации и соседние гиперпараметры

- Сделаем начальные условия, но заменим одну функцию активации на сигмоиду.



**Рисунок 18** – Два слоя у полносвязной нейросети, 60 эпох, Sigmoid. Справа Test Loss, слева Train Loss

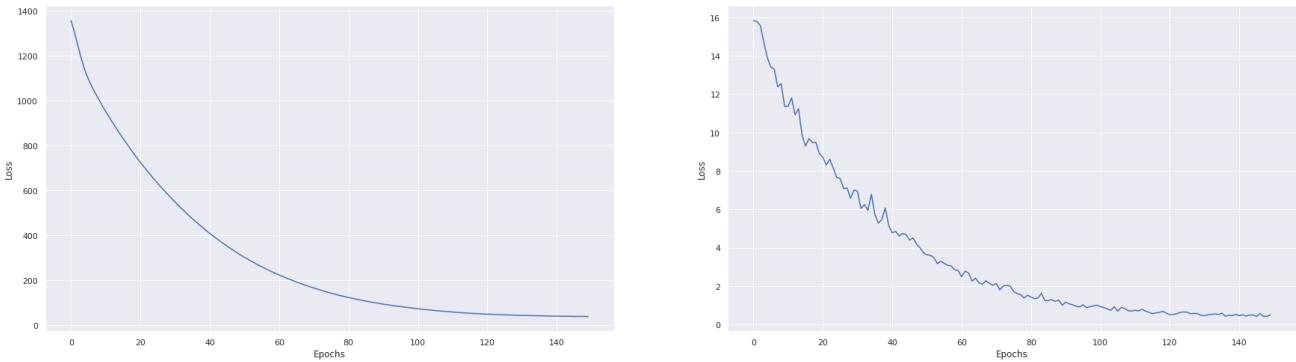
- Увеличим количество слоев до трех.



**Рисунок 19** – Три слоя у полносвязной нейросети, 60 эпох, Sigmoid. Справа Test Loss, слева Train Loss

- Увеличим количество эпох до 150.

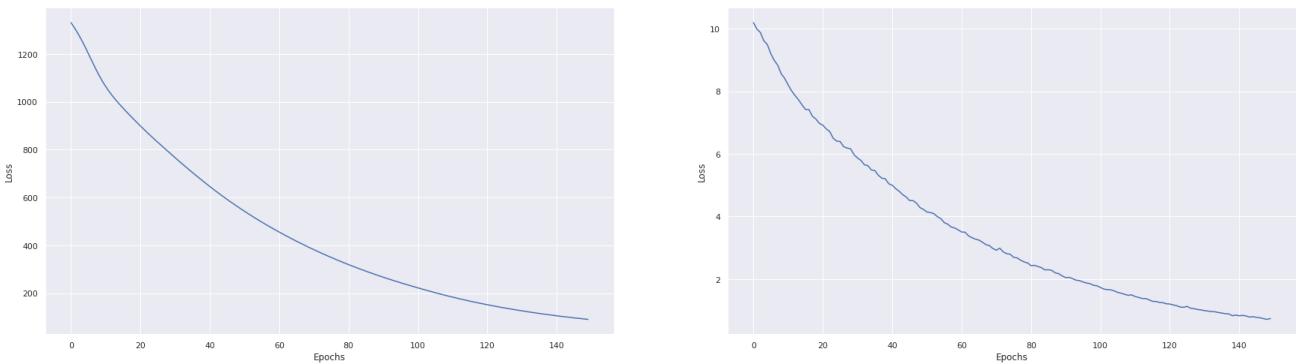
Можно увидеть, что сигмоида показала себя хуже, чем гиперболический тангенс, однако можно попробовать увеличить batch size (начальное значение = 100).



**Рисунок 20** – Три слоя у полносвязной нейросети, 150 эпох, Sigmoid. Справа Test Loss, слева Train Loss

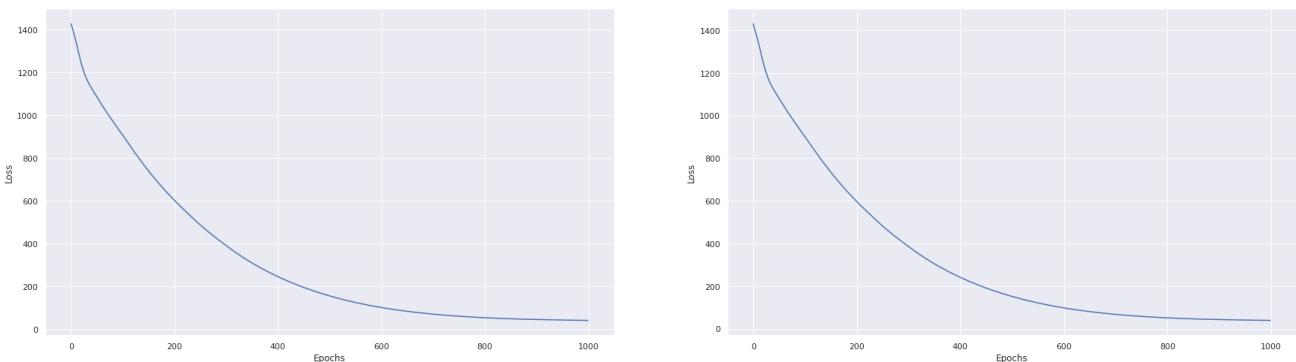
### 5.1.3. Изменим batch size

1. Увеличим «batch size» в 2 раза.



**Рисунок 21** – Три слоя у полносвязной нейросети, 150 эпох, Sigmoid, batch size=200. Справа Test Loss, слева Train Loss

2. Увеличим «batch size» еще в 5 раз.



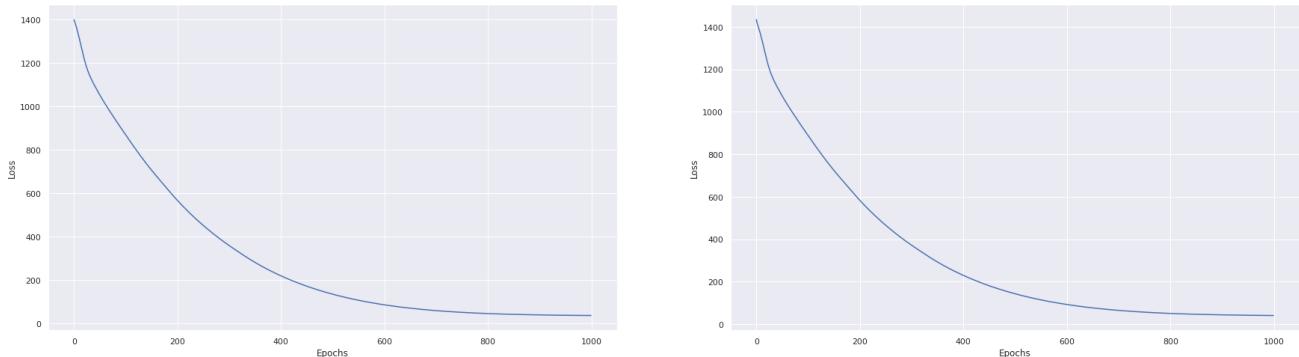
**Рисунок 22** – Три слоя у полносвязной нейросети, 1000 эпох, Sigmoid, batch size=1000. Справа Test Loss, слева Train Loss

Как мы видим, увеличение batch size тормозит обучение нейросети, что требует увеличение количества эпох, но убирает «артефакты», которые получаются при обучении полносвязной НС при помощи сигмоидальной функции активации.

#### 5.1.4. Изменим процент тестовой выборки

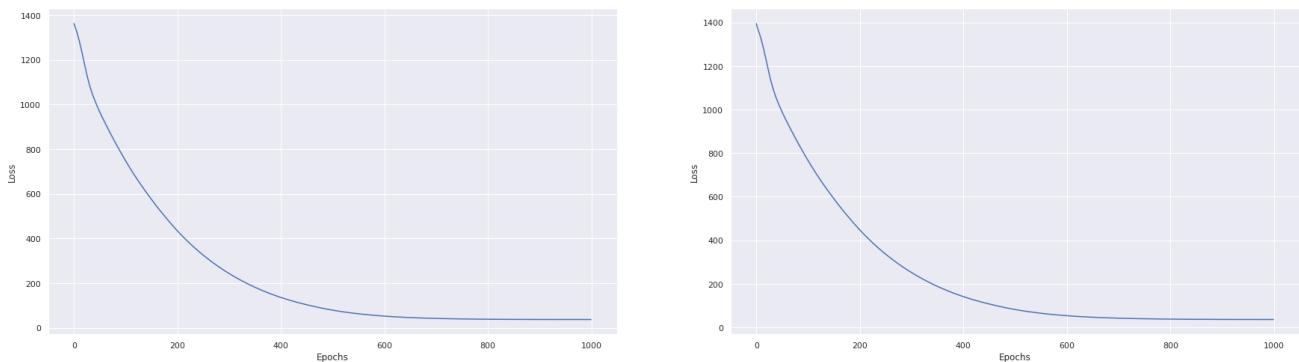
Теперь попробуем уменьшить выборку «train» и увеличить % тестовой выборки. Будем использовать нейросеть с параметрами из пункта выше.

1. Процент тестовой выборки: 50.



**Рисунок 23** – Три слоя у полносвязной нейросети, 1000 эпох, Sigmoid, batch size=1000, test size=0.5. Справа Test Loss, слева Train Loss

2. Процент тестовой выборки: 75.



**Рисунок 24** – Три слоя у полносвязной нейросети, 1000 эпох, Sigmoid, batch size=1000, test size=0.75, dropout. Справа Test Loss, слева Train Loss

При увеличении тестовой выборки и применении дропаута нейросеть себя достаточно хорошо ведет и ухудшает распознавание клеток совсем на немного.

## 5.2. Реализация U-Net

Эта нейросеть потребовала от нас намного большего терпения и больших усилий, чтобы получить какие-то адекватные результаты.

Мы подготовили несколько выборок, чтобы понять, с какими условиями и трудностями может справиться наша НС:

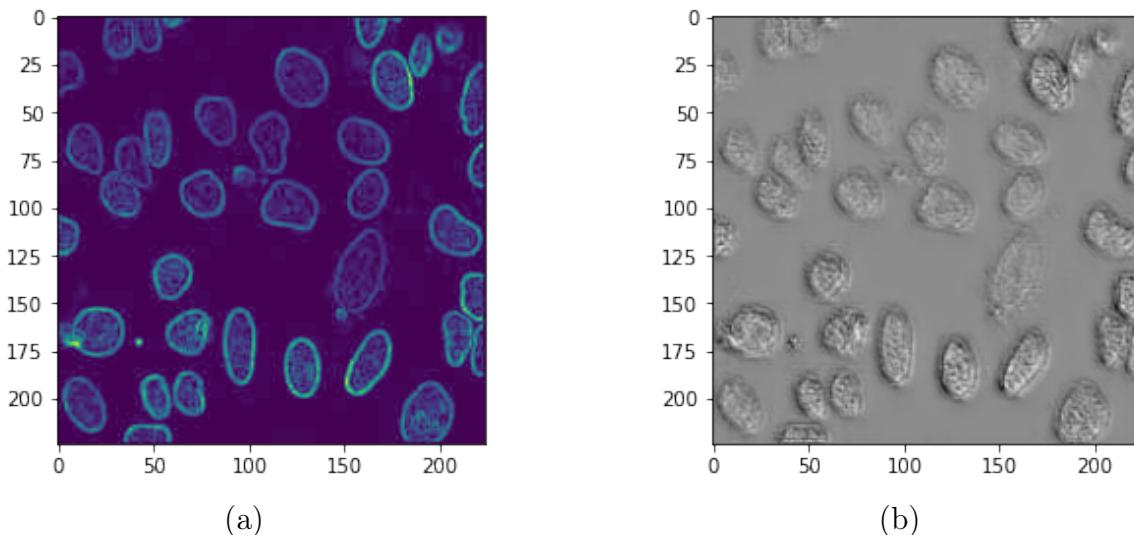
1. Выборка из 120 картинок, где нет шумов, только поворот каждой картинки на полный оборот по 30 градусов.
2. Выборка из 400 картинок, где нет чистых изображений, только картинки с 10 различными шумами и 4 поворотами.

3. Выборка из 720 картинок, где есть чистые картинки, повороты по 30 градусов и 5 различных шумов.
4. Выборка из 1340 картинок, где есть чистые картинки, повороты по 30 градусов и 10 различных шумов.

Рассмотрим первую, самую простую, выборку.

### 5.2.1. Рассмотрим первую, самую простую, выборку

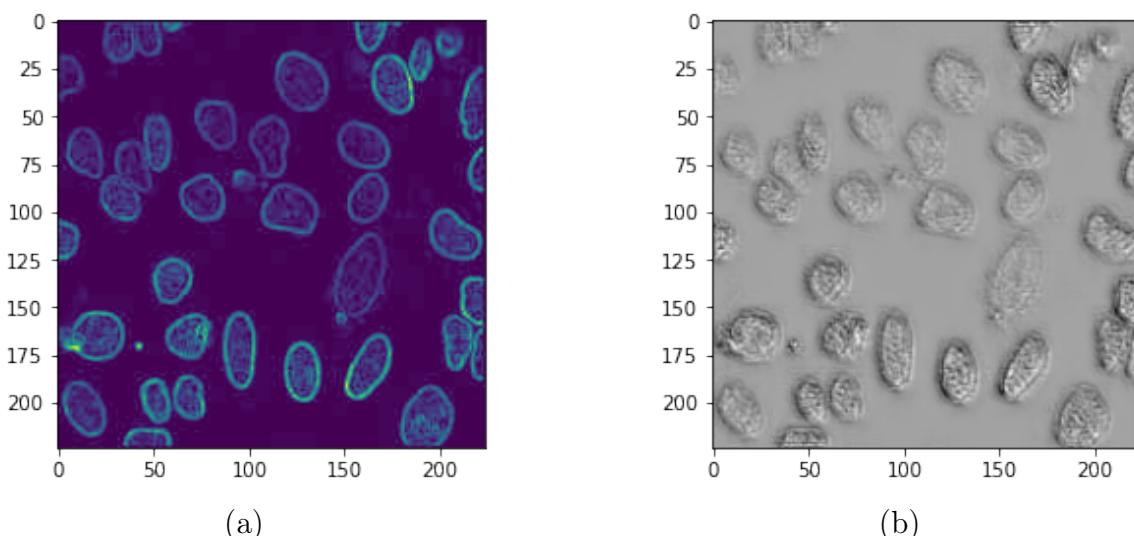
1. Начнем с описания параметров нейросети. Мы задаем  $batch\ size = 1$ ,  $learning\ rate = 10^{-4}$ ,  $num\ epochs = 10$ .



**Рисунок 25** – а) Созданная маска изображения; б) Предсказание нейросети на первой эпохе

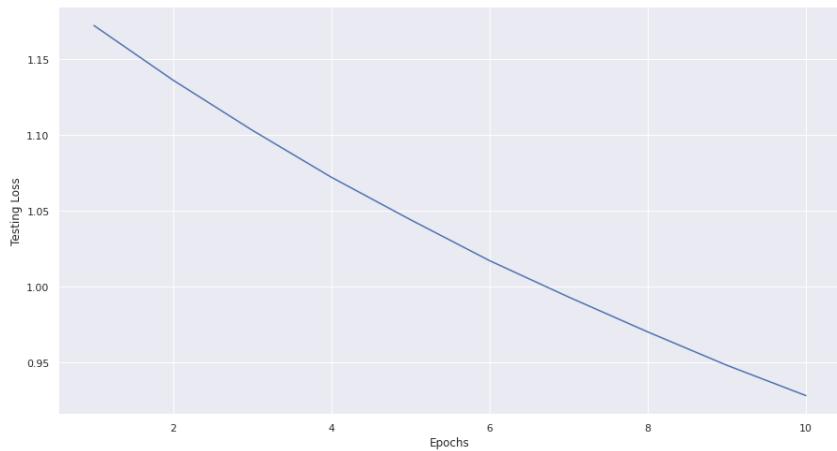
На первой эпохе  $testing\ loss = 1.172$ .

Сравним с результатом на последней эпохе:



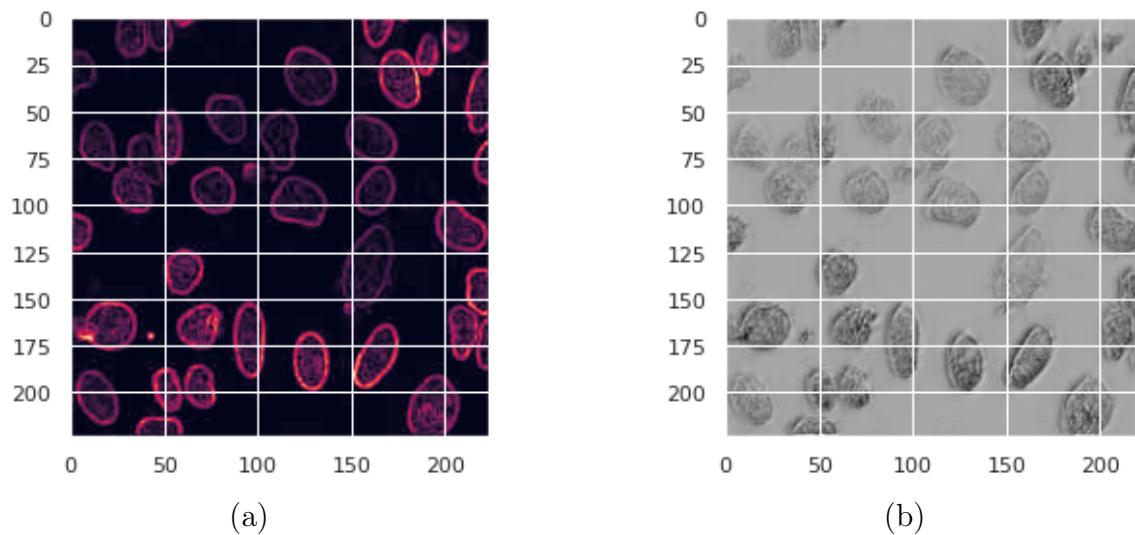
**Рисунок 26** – а) Созданная маска изображения; б) Предсказание нейросети на последней эпохе

Построим график зависимости  $testing\ loss$  от номера эпохи:



**Рисунок 27** – График зависимости Testing Loss от номера эпохи

2. Увеличим *learning rate* в 10 раз, чтобы посмотреть на результат



**Рисунок 28** – Увеличенный в 10 раз *learning rate*: а) Созданная маска изображения; б) Предсказание нейросети на первой эпохе

На первой эпохе *testing loss* = 0.696.

### 5.2.2. Посмотрим другие выборки

Посмотрим на результаты работы НС на двух других выборках:

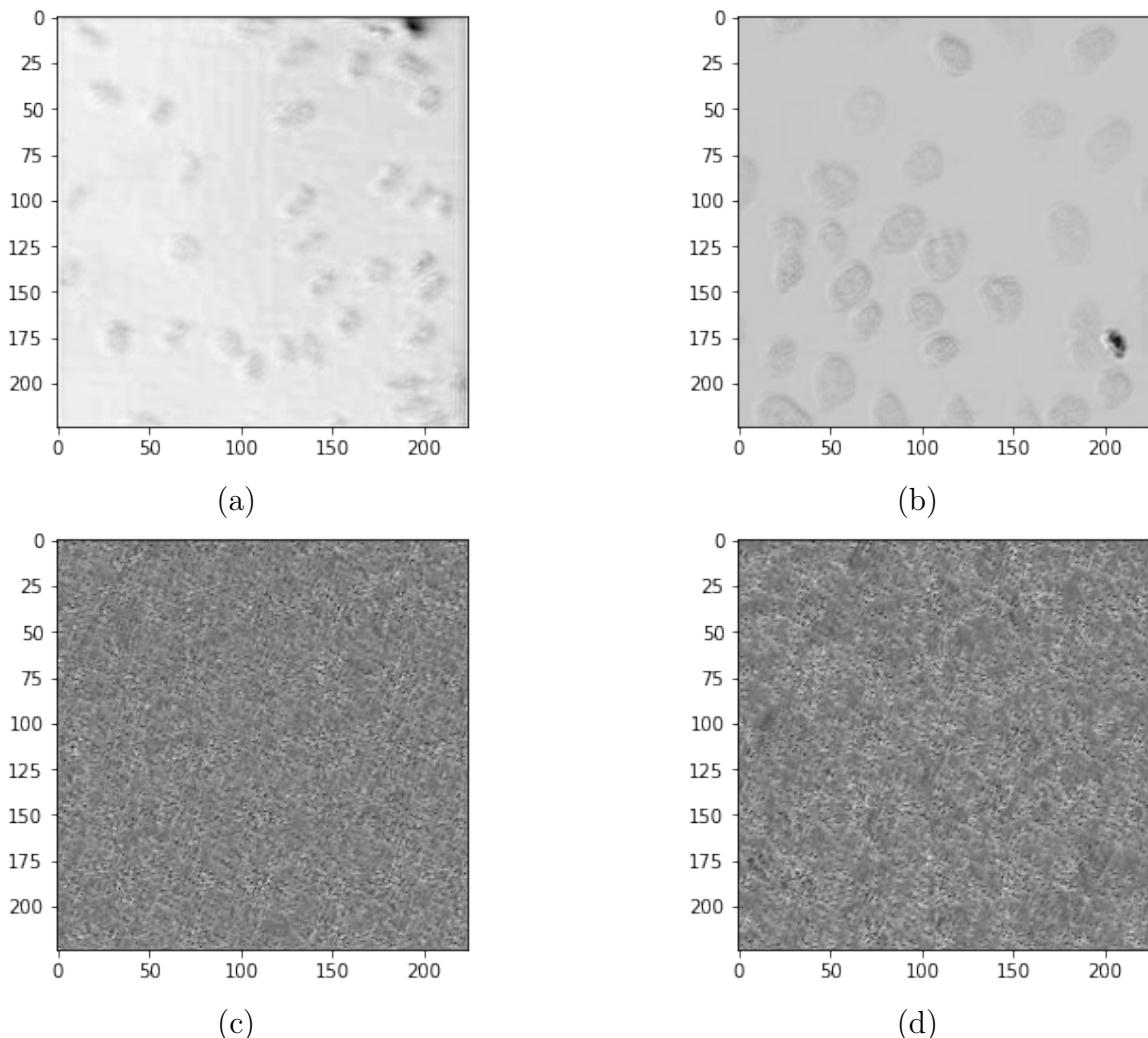
### 5.2.3. Поменяем оптимизаторы программы

Представим сводную таблицу оптимизаторов и *Testing Loss*:

Отсюда можно сделать вывод, что наш выбор оптимизатора в виде «Adadelta» был не спонтанным, а сторого обоснован с научной точки зрения.

### 5.2.4. Поменяем loss function программы

Теперь зафиксируем наш оптимизатор и будем менять loss function. До этого во всех опытах мы использовали «HingeEmbeddingLoss», так как он показал себя лучше остальных.



**Рисунок 29** – Результаты выборок (сверху – 720, снизу – 1340: а) Первая эпоха; б) Третья эпоха; в) Первая эпоха; г) Пятая эпоха

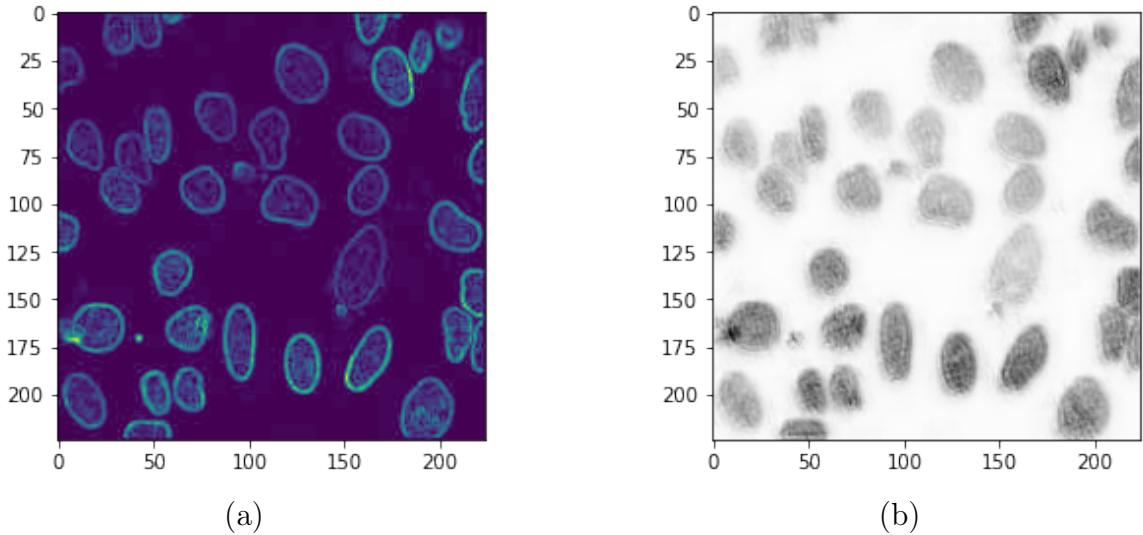
**Таблица 1** – Варьирование оптимизатора

Optimizer	Testing Loss
Adam	9850
SGD	0.227
Rprop	21480
AdamW	5208
RMSprop	6141298688
Adamax	26

**Таблица 2** – Варьирование loss function

Loss Function	Testing Loss
MSELoss	37244
L1Loss	148
KLDivLoss	665

Здесь приведена достаточно не информативная информация, так как надо было отслеживать изменение Testing Loss по эпохам. Мы сделали эту операцию за кадром, и она была стерта прежде, чем появились какие-то зерна нашего отчета, однако это отложилось



**Рисунок 30** – Пример работы оптимизатора SGD: а) Созданная маска изображения; б) Предсказание нейросети на первой эпохе

в нашей памяти, из чего мы выбрали именно «HingeEmbeddingLoss»

Можно сделать вывод, что мы неплохо обучили нейросеть «видеть» и рисовать изображения клеток. Обучение происходит правильно, так как с каждой эпохой Testing Loss падает.

### 5.3. Реализация предобученных нейросетей

Попробуем увидеть какая из уже перечисленных сетей лучше подходит для нашей задачи, и какие оптимальные параметры ей соответствуют. Будем использовать для этого предобученные сети из модуля `torchvision.models`, варьируя при этом следующие параметры: оптимизатор сети, loss function, размер batch.

На вход сети подаётся трёхканальное изображение с клетками, на выходе из сетки ожидается число клеток на изображении, оно сравнивается с заранее просчитанным количеством. Также в течение работы сетки записывается информация о непосредственной разнице между предсказаниями сети и реальными значениями, а также динамика изменения loss function на тестовой и тренировочной выборках. Отбирается сеть с минимальными потерями на тестовой выборке и сохраняются значения её весов.

Далее приведены примеры наиболее успешных испробованных вариаций.

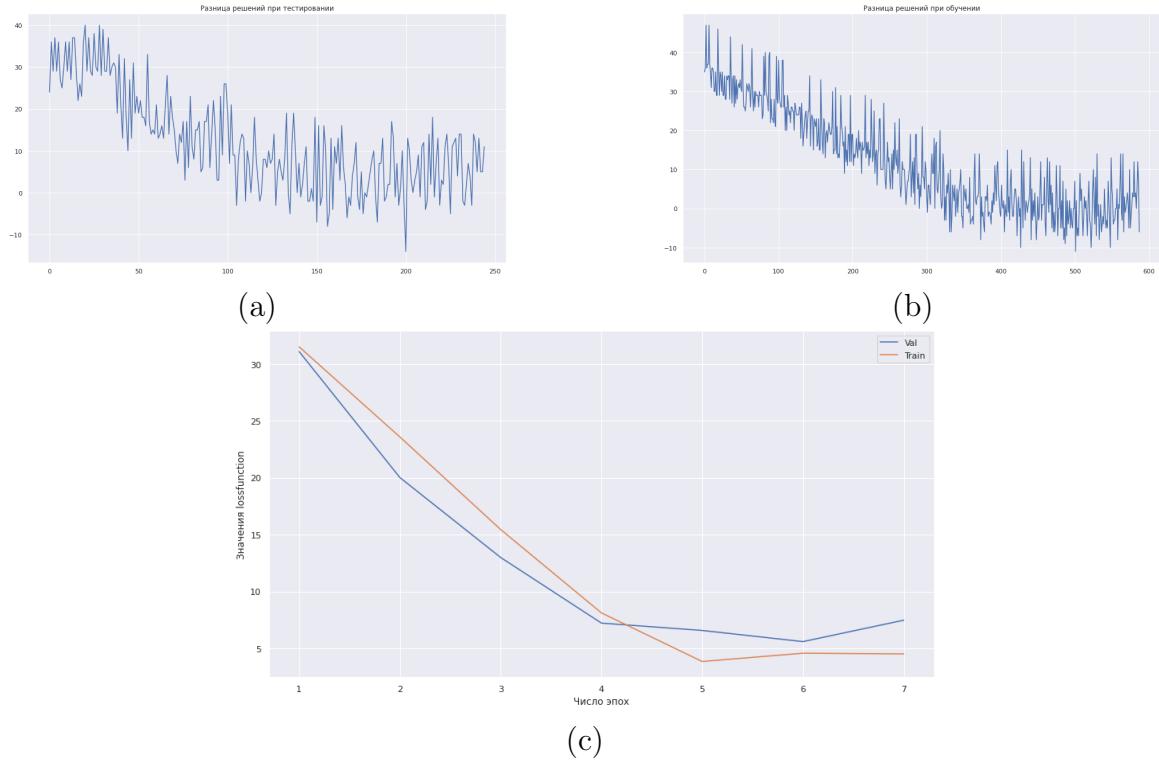
Наиболее оптимальным образом показали себя сети VVG, ResNet и DenseNet, особенно оптимальной кажется сеть ResNet не только из-за точности предсказаний, но и скорости предсказаний.

В роли оптимизаторов оказались подходящими такие функции как SGD и Adam со схожими показателями при разных комбинациях основных параметров.

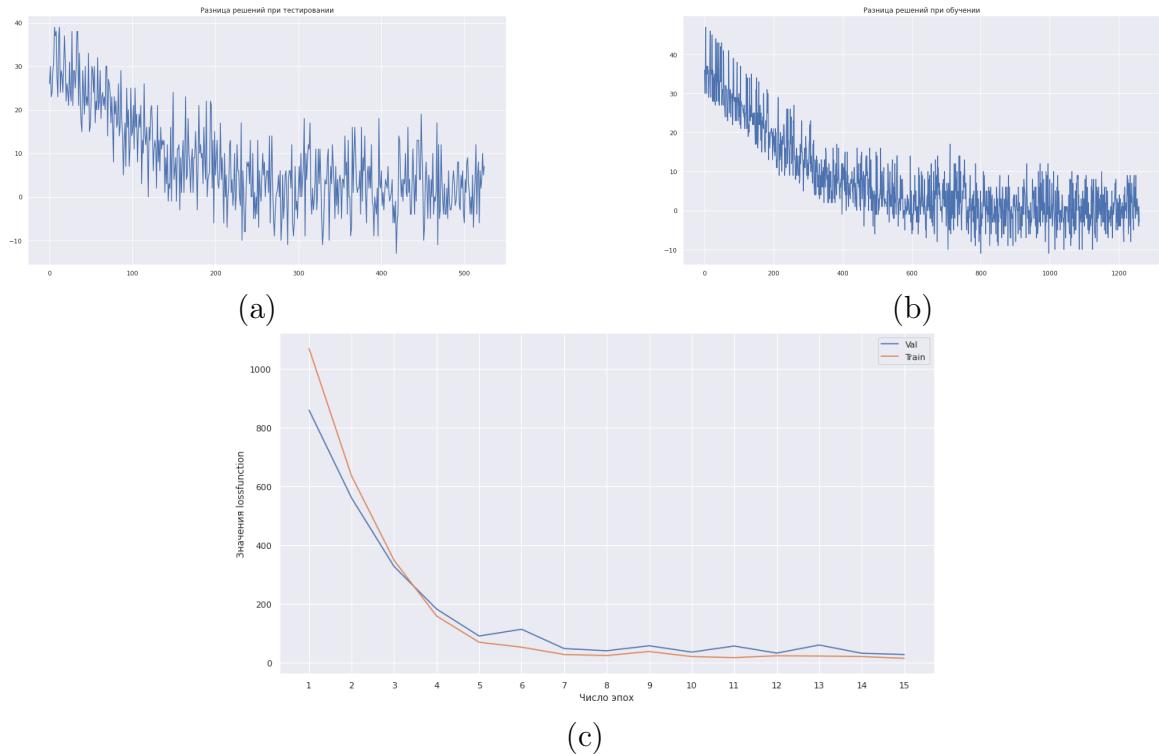
Выбор loss function был ограничен L1, MSE и SmoothL1 так как нам интересно более точное количество объектов выбранного класса, а не количество классов на изображении.

Как видно из всех графиков loss function сеть достигает пика своего обучения уже на 4-10 эпохах, а затем не вносится значительных улучшений, а в некоторых случаях даже идёт ухудшение модели. Для решения этой проблемы было испробован метод изменения learning rate при помощи lr scheduler. Однако хотя колебания loss function после пика обучения и уменьшились, но на улучшения оптимума модели это не повлияло.

Также надо отметить изменения возникающие при перемене размера batch. Наиболее используемые в работе размеры 5 и 10. При переходе на размер 10 от 5 изменения формы

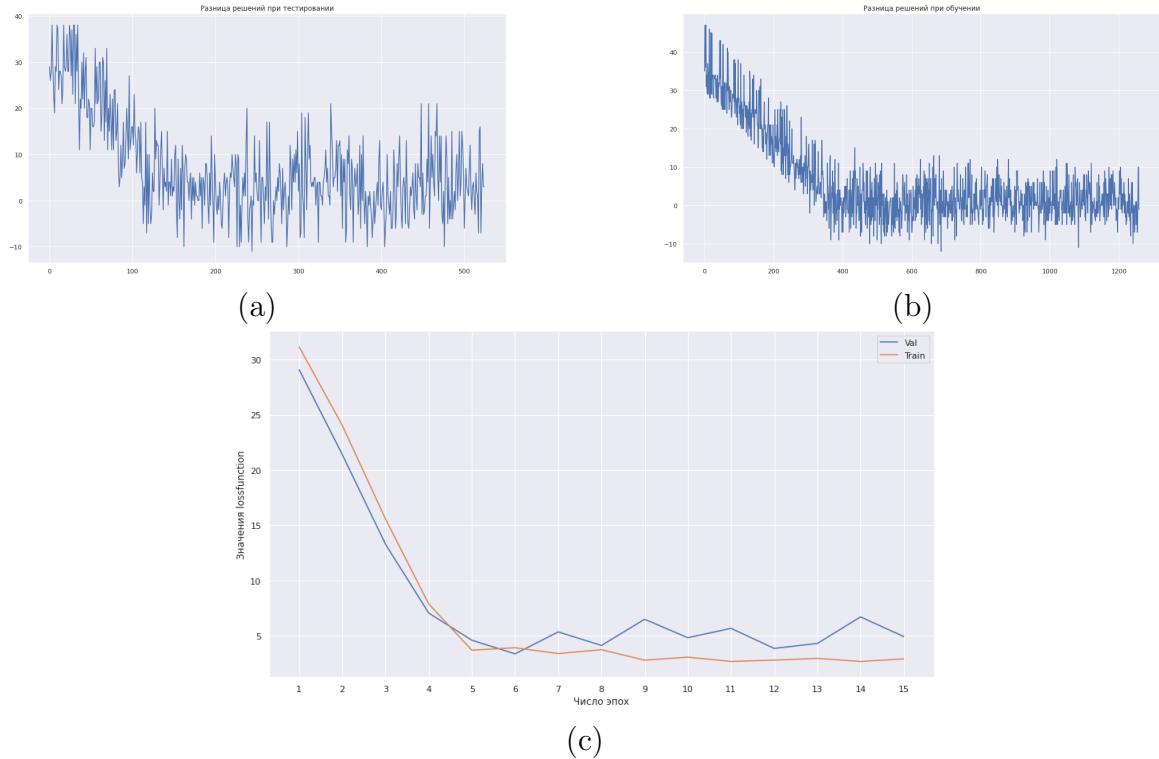


**Рисунок 31** – Модель: DenseNet, оптимизатор: Adam, loss function: L1, batch : 5; a) Разница на тестовой выборке; b) Разница на тренировочной выборке; c) Графики loss function тестовой и тренировочной выборки

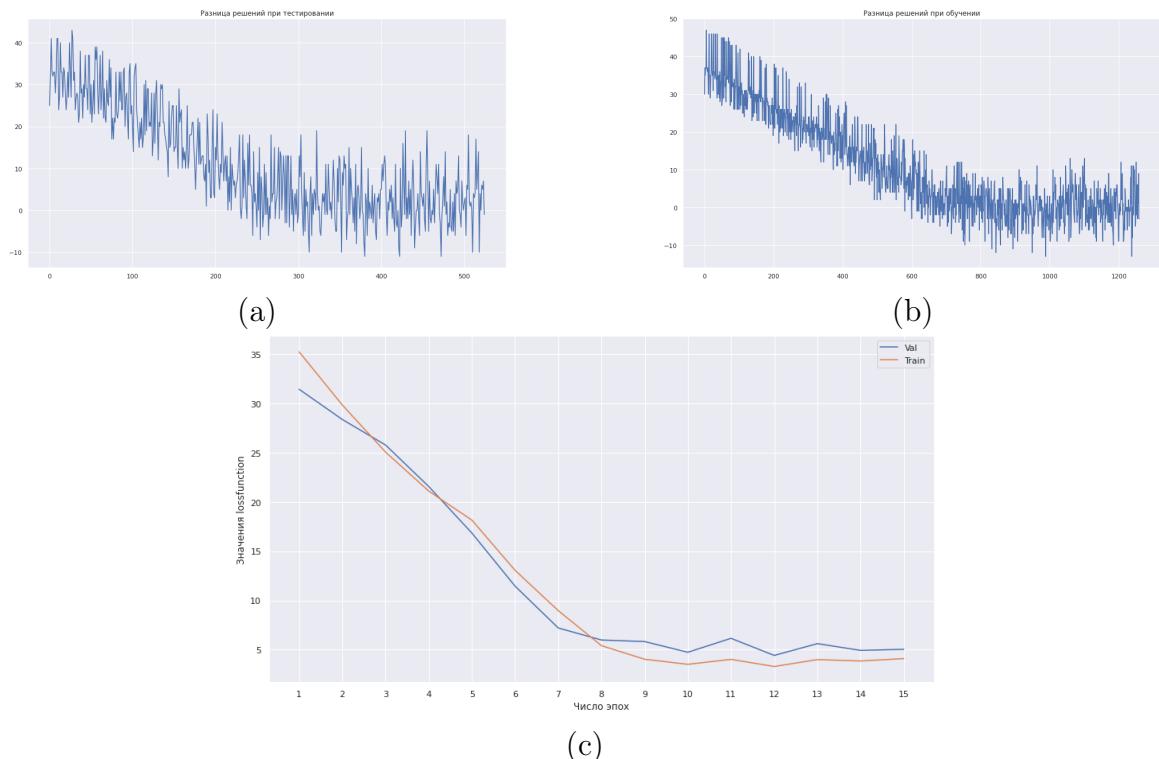


**Рисунок 32** – Модель: ResNet, оптимизатор: Adam, loss function: MSE, batch : 5; a) Разница на тестовой выборке; b) Разница на тренировочной выборке; c) Графики loss function тестовой и тренировочной выборки

графика loss function не наблюдалось, однако скорость обучения снижалась, что иногда

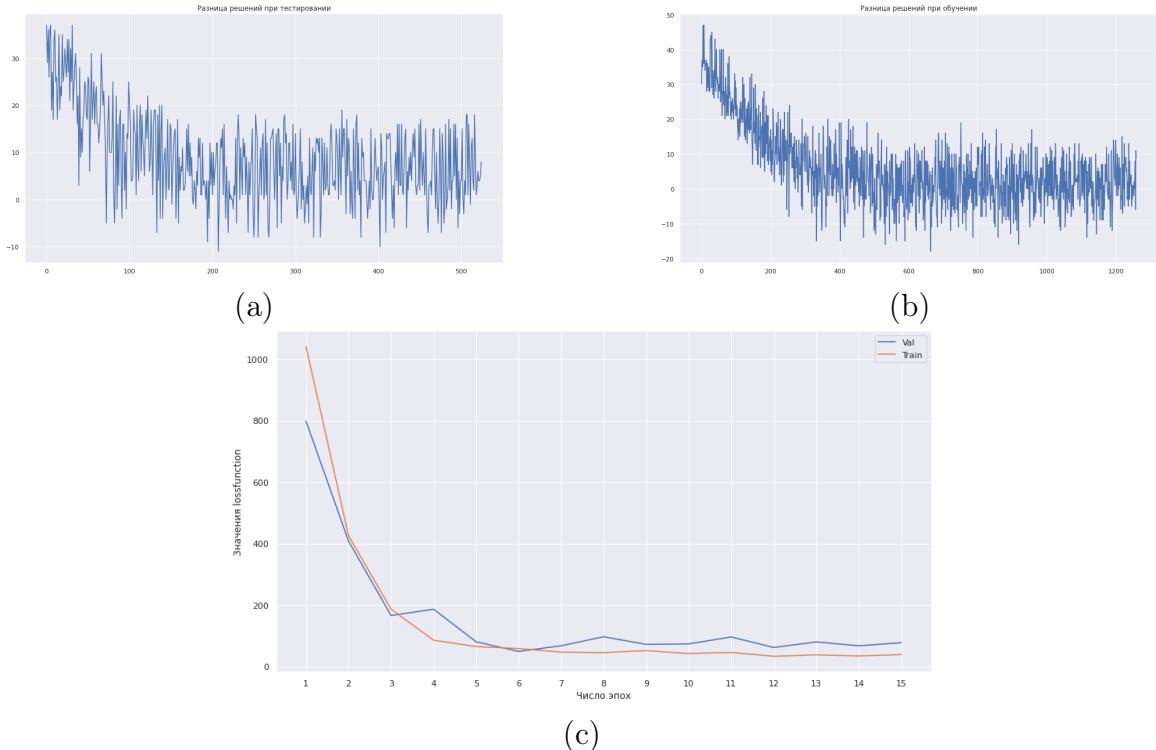


**Рисунок 33** – Модель: DenseNet, оптимизатор: Adam, loss function:SmoothL1, batch : 5; а) Разница на тестовой выборке; б) Разница на тренировочной выборке; в) Графики loss function тестовой и тренировочной выборки

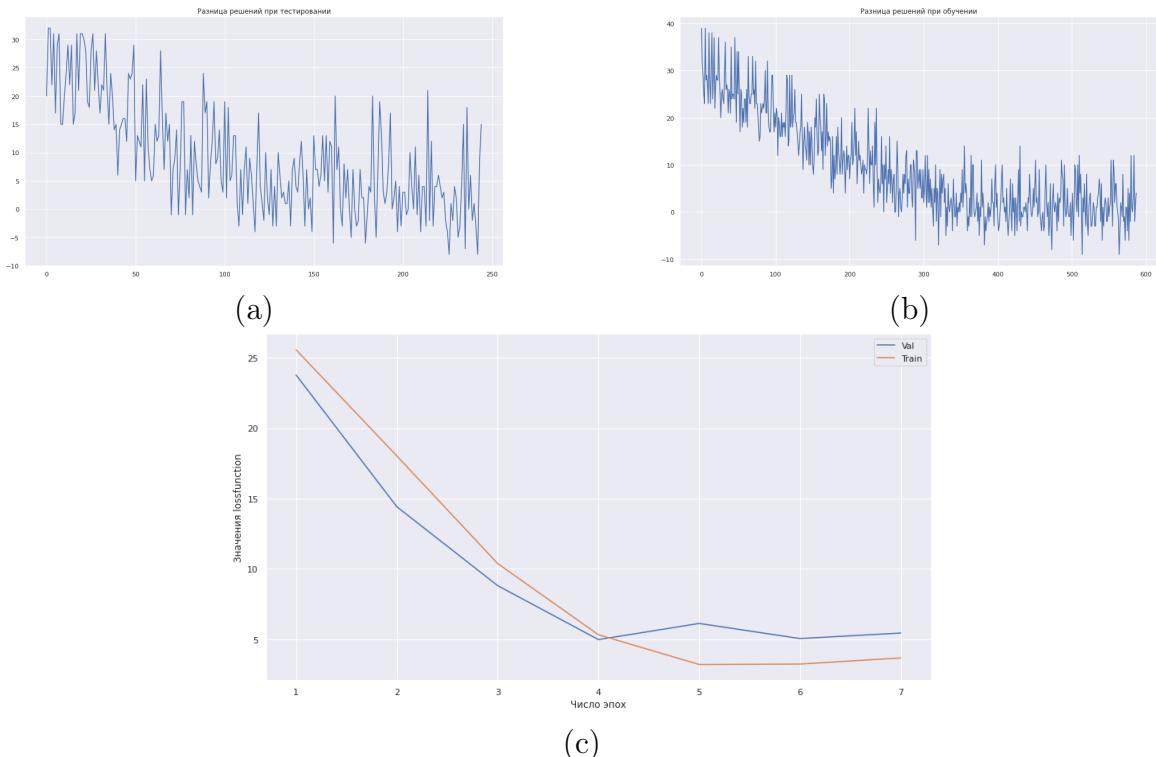


**Рисунок 34** – Модель: ResNet, оптимизатор: Adam, loss function:SmoothL1, batch : 10; а) Разница на тестовой выборке; б) Разница на тренировочной выборке; в) Графики loss function тестовой и тренировочной выборки

позволяло немного улучшить точность метода.



**Рисунок 35** – Модель: VGG, оптимизатор: Adam, loss function: MSE, batch : 5; а) Разница на тестовой выборке; б) Разница на тренировочной выборке; в) Графики loss function тестовой и тренировочной выборки



**Рисунок 36** – Модель: ResNet, оптимизатор: SGD, loss function: L1, batch : 5; а) Разница на тестовой выборке; б) Разница на тренировочной выборке; в) Графики loss function тестовой и тренировочной выборки

## 6. Обсуждение

Нами были созданы три нейронных сети, которые выполняют разные задачи: две из них определяют количество клеток на изображении, а одна выделяет клетки на изображении.

К сожалению, мы были технически ограничены производительностью имеющихся в нашем распоряжении вычислительных устройств, поэтому часть запланированных улучшений будут добавлены уже в дальнейшем развитие проекта. Каждый прогон нашей нейронной сети на выборке для 10 эпох тратил от 3 часов до 9 (в зависимости от даваемых параметров), что вынуждало нас сконцентрироваться на улучшении уже достигнутых результатов и не позволило далеко продвинуться в усложнении нашего проекта.

Мы считаем, что при наличии времени и более производительных аппаратов, мы могли бы улучшить наш проект, проведя дополнительные исследования по увеличению выборки и количеству эпох. В дальнейшем развитие проекта мы видим следующие возможные идеи для повышения качества определения числа клеток. Первое, возможно непосредственное прямое совмещение второй и третьей нейросетей, это направление может оказаться продуктивным так как уже выделенные на изображении клетки должны быть более лёгкой мишенью для определения их числа последующей нейросетью. Второе, конструирование гибридной конструкции из двух сетей, так, чтобы отсчёт клеток производился не по исходной картине или обработанному изображению, а по внутреннему представлению второй нейросети этого изображения.

Перспективы этой программы заключаются в том, что при подборе оптимальных параметров, можно достаточно быстро и точно автоматизированно получать количество клеток на изображении. Например, эту систему можно обучить распознавать мертвые клетки и живые, что очень полезно в работе любого клеточного биолога, когда вместо приблизительной оценки глазами или ручного анализа каждой картинки, изображения можно подать на вход нейросети, чтобы получить адекватный и достаточно точный результат.

Так же, благодаря гибкой системе создания масок для выборки, распознавать можно практически любые элементы при должном качестве снимка, светимость и местоположение. Код на Python3 удобен тем, что можно автоматически не только определить нужный параметр, но и добавить его в таблицу или базу данных, что существенно ускоряет работу.

С помощью такой НС можно определять размер клеток/ядер, что позволит найти, например, какую-то патологию или аномалию в ткани или колонии клеток. Это поможет наблюдать и контролировать все внутренние процессы клетки, что многократно упрощает работу ученых, которые могут положиться на «машину» ради достижения желанного результата.

Можно придумать бесконечное количество применений такой нейросети, так как она легко модифицируется, усложняется, при этом не теряя своей точности и надежности.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- [1] Rosenblatt F. The perceptron: A plobalistic model for information storage and organization in the brain // Psychological Review. 1958. Vol. 65. No. 6. P. 386.
- [2] Restak R. M., Grubin D. The Secret Life of the Brain. Joseph Henry Press, 2001.
- [3] Mc.Culloch W. S., Pitts W. A logical calculus of the ideas immanent in nervous activity // The Bulletin of Mathematical Biophysics. 1943. Vol. 5. No. 4. Pp. 115-133.
- [4] Nair V., Hinton G. E. Rectified Linear Units Improve Restricted Boltzmann Machines // Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010.
- [5] Rosenbloom P. The method of steepest descent // Proceeding of Symposia in Applied Mathematics. 1956. Vol. 6.
- [6] Tikhonov A. N., Glasko V. B. Use of the regularization method in nonlinear problems // USSR Computational Mathematics and Mathematical Physics. 1965. Vol. 5. No. 3. Pp. 93-107.
- [7] Srivatstava N. et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting // Journal of Machine Learning Research. 2014. Vol. 15. No. 1. Pp. 1929-1958.
- [8] Cohen A. I. Rodes and Cones // Physiology of Photoreceptor Organs. Springer Berlin Heidelberg, 1972. Pp. 63-110.
- [9] Viola P., Jones M. Rapid Object Detection using a Boosted Cascade of Simple Features // Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. Vol. 1. IEEE, 2001.