

**ОТЧЕТ ПО ВЫПОЛНЕННОЙ РАБОТЕ В РАМКАХ АЛЬТЕРНАТИВНОГО ЭКЗАМЕНА МЛиТА-2023**

Тема:

“Сопровождение объекта по видео”

| Студентки гр. 1308 ФКТИ |  | Михайлова М. А. |
| --- | --- | --- |
|  |  | Морик А. И. |
|  |  | Черникова П. В. |
| Научный руководитель |  | Филатов А. Ю. |

Санкт-Петербург

2023 г.

**Цель:**

Разработка трекера, чья задача отслеживать любой заданный пользователем объект. Трекер должен быть устойчив к исчезновениям объекта из кадра, а также уметь выводить вероятность, с которой найдена рамка, где рамка – прямоугольник, которым отмечается отслеживаемый объект.

**Задача:**

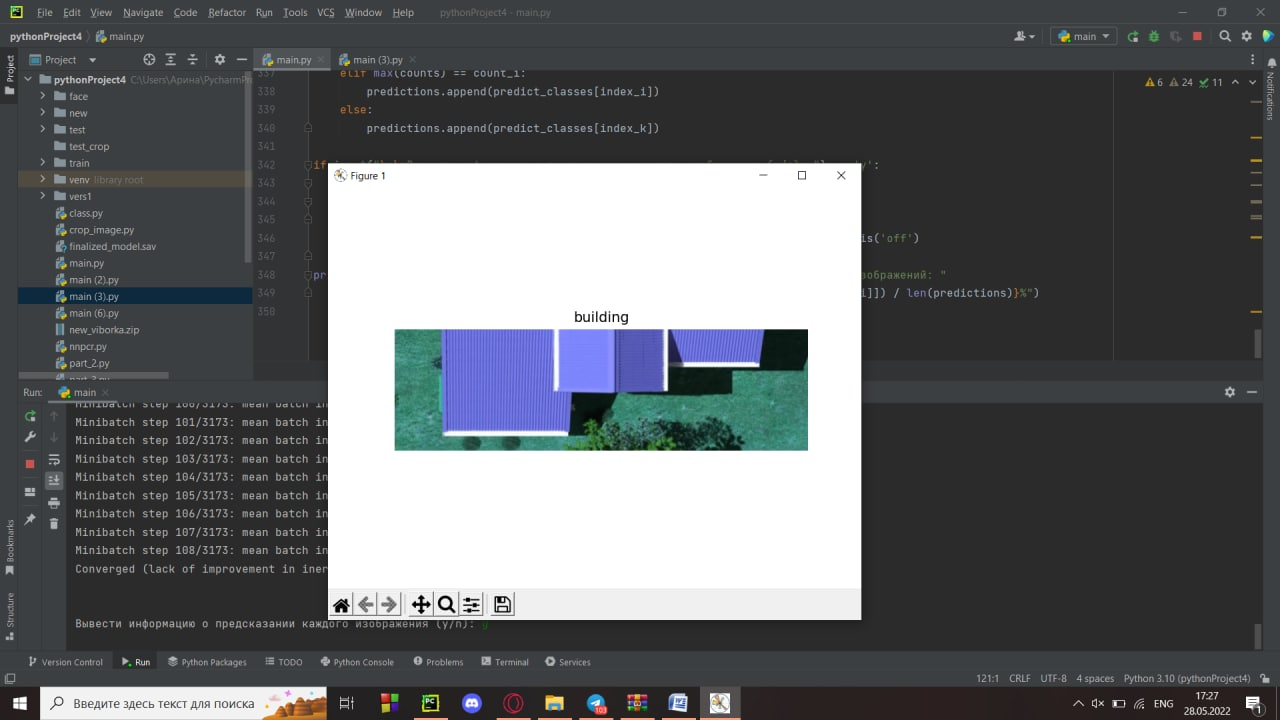
Написать нейросеть, удовлетворяющую заданным условиям.

**Введение**

Данная работа является продолжением череды проектов, объединенных задачей детекции объектов на изображениях с дрона.

Детекция – это задача, подразумевающая поиск на изображении координатов ограничивающей рамки объекта и классификация оного. Так, в весеннем семестре прошлого года мы разработали программу, использующую машинное обучение: на вход подается датасет, составленный из обрезанных фотографий нескольких классов (мы работали с классами “Машина”, “Здание” и “Дерево”, но это не имеет особого значения, если будет составлен аналогичный датасет для других классов), на выходе демонстрируются предсказания для переданных изображений.

Наш детектор использовал два способа извлечения из изображения отличительных признаков: нахождение дескрипторов и цветной гистограммы для каждого изображения. Полученные результаты были использованы для обучения алгоритмов kNN и K-means, позже объединенных в ансамбль. Для создания детектора также были применены некоторые другие технологии – о них и об алгоритмах, упомянутых выше, подробнее можно прочесть в [отчете.](https://drive.google.com/file/d/13_f18PAadw7KVhrBeYcHixxrzwz6g6xf/view?usp=sharing) Там же можно найти ссылку на репозиторий GitHub с подробно прокомментированным кодом.

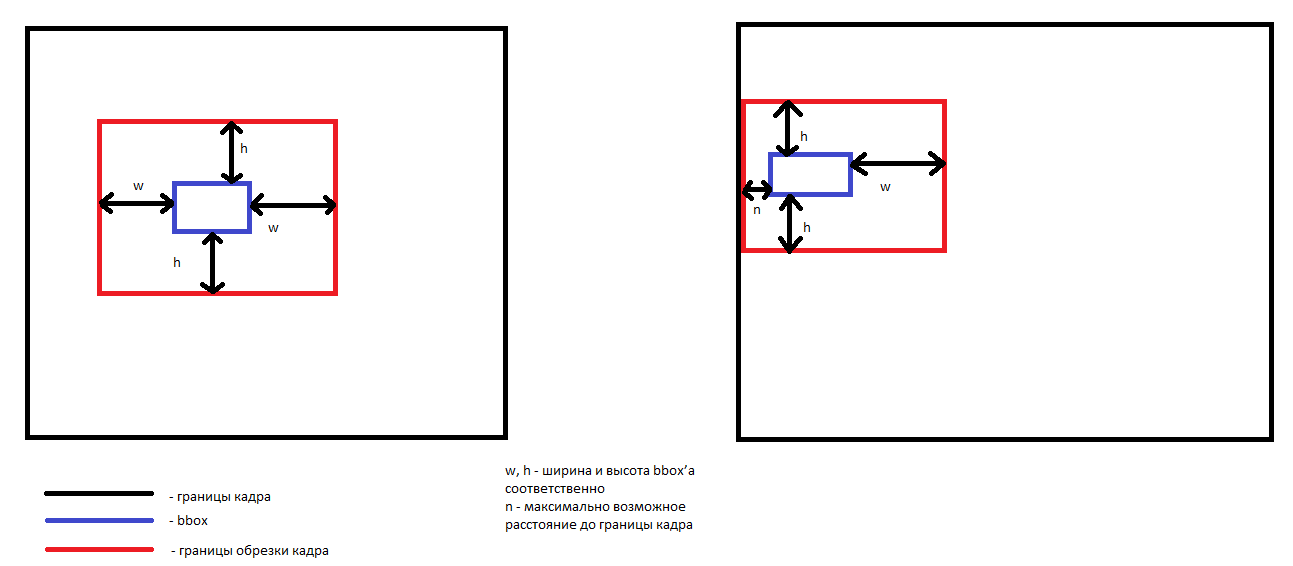


*рис. 1. Пример предсказания разработанным детектором.*

Полученный в результате детектор имеет 83% точности, измеренные на 30 тестовых изображениях различных классов и разного процента нахождения самого объекта в кадре (см. рис. 1 – на изображении находится лишь часть дома). Детектор обучался на 88 изображениях, датасет был собран вручную.

В следующем семестре задача была усложнена, и мы начали работу над трекером. На этот раз необходимо было не только найти объект, но и отследить его перемещение так, чтобы он постоянно находился в рамке. То есть, предсказание теперь строится не только на каких-то внешних отличительных признаках отслеживаемого объекта, но и на его динамике движения в кадре.

Были изучены некоторые уже существующие решения. Так, мы подробно разобрали популярную нейросеть-детектор YOLO в нескольких ее версиях, а также алгоритм трекинга SORT, также рассмотренный еще и в вариации DeepSORT. В процессе разбора теоретической части, также была проделана работа над практической – мы пробовали различные комбинации YOLO и SORT. После сравнений была выбрана связка DeepSORT и YOLOv5, а также разработан и применен алгоритм обрезки. Обрезка была необходима для увеличения точности результата, ее действие описано на следующей схеме:



*рис. 2. Условная схема обрезки.*

где *bbox* – ограничительная рамка, обнаруженная в кадре. Координаты найденной границы передаются для следующего кадра, и именно по ним он обрезается, что позволяет трекеру “сосредоточиться” на конкретной области изображения вместо того, чтобы проверять всю предоставленную площадь. Соответственно, координаты обрезки обновляются с каждым найденным в кадре объектом, т.к. предполагается, что объект движется.

Для того, чтобы понять актуальность обрезки для нашей задачи, достаточно рассмотреть следующую составленную нами таблицу:

*Табл. 1. Сравнительная характеристика работы трекера до добавления обрезки и после.*

| Обработка видео | Результат нахождения объекта | Время обработки одного кадра без выбросов, с |
| --- | --- | --- |
| Без обрезки | 53% (для видео с 1464 кадрами, где объект ни разу не пропадал из кадра) | 0.700-0.850 |
| С обрезкой | 91% | 0.450-0.550 |

Наглядный результат работы представлен на [видео](https://drive.google.com/file/d/1WOhWJSIinKsmy6DNgUOwL1dCH8iwa2Qo/view?usp=sharing), а подробности об использованных алгоритмах, процессе сравнения и прочих деталях можно найти в [отчете](https://docs.google.com/document/u/0/d/1BtbloC-ZMJsoeYMfQ-kEu9nxhaFWzFf_GjQqFTOHDmA/edit).

При планировании работы на этот семестр мы решили немного расширить “пул” решаемых нами задач. В силу того, что мы уже имеем опыт работы с нейронными сетями, нам стало интересно разработать собственную. Также, это решение обусловлено тем, что используемый нами в прошлой работе YOLO ориентируется на уже существующие классы, на которых он был обучен, однако для нашей задачи будет куда правильнее не использовать классы вообще. Это значит, что детектор, по сути, будет выполнять бинарную классификацию, то есть, определять: тот объект в кадре, который был задан изначально, или нет. Именно такую технологию мы и хотели разработать.

Для этого нам предстояло сначала завершить еще одну задачу: просмотреть уже предложенные кем-то решения, изучить архитектуры нейросетей, которые могли бы лечь в основу нашей работы, подробно разобрать теоретическую часть. Это работа довольно обширная, ведь одно дело – пользоваться готовой нейросетью, рассматривая ее внутренности с точки зрения разработчика, пользующегося ей, а другое – провести сравнительную характеристику архитектур, выбрать одну из них по определенным критериям, разобрать ее слои и подготовить почву для написания кода. В следующих ниже разделах даны ответы на все поставленные вопросы и задачи.

**Содержание**

[Что такое сиамские нейросети? 5](#_p2f05w96dh3f)

[Обработка изображений 7](#_v6sip44xkjko)

[Используемая функция активации: ReLU 8](#_4puzban1qe1e)

[Нормализация 9](#_gjnyk1kn0nkv)

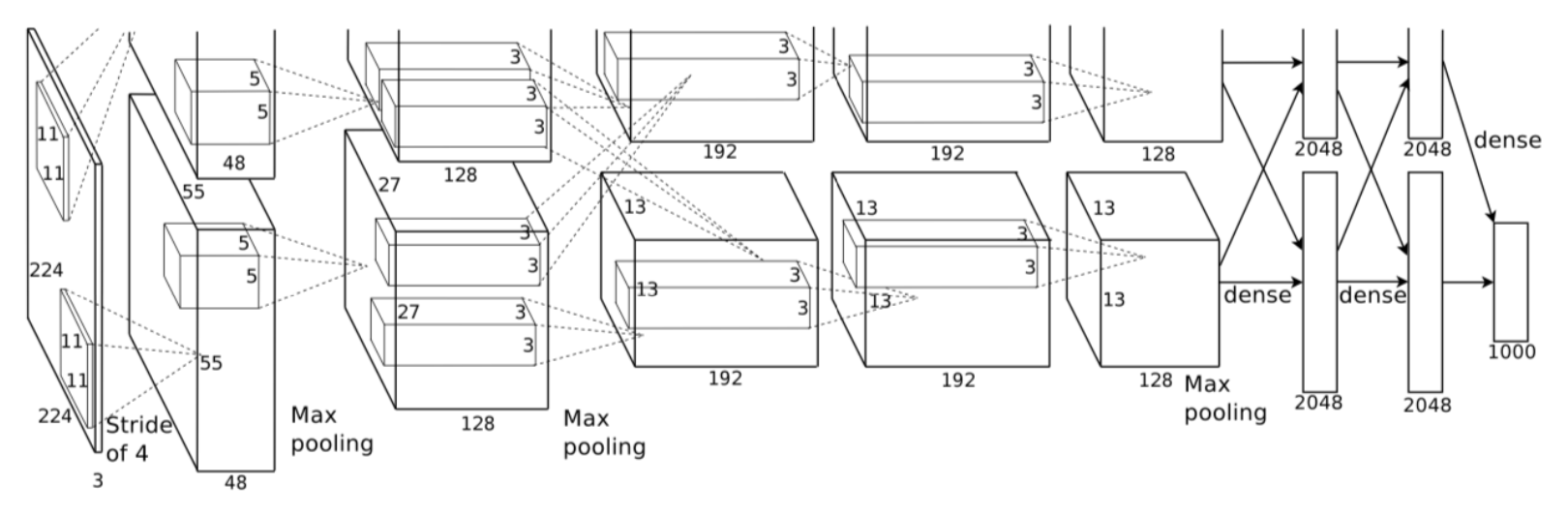
[Регуляризация нейросети: Dropout 12](#_jhigwgnbq5gj)

[Архитектура нейронной сети 14](#_sj7izhuutxen)

# 

# **Что такое сиамские нейросети?**

*Сиамская нейронная сеть* (англ. Siamese neural network) — это разновидность искусственной нейронной сети, которая состоит из двух идентичных нейронных подсетей с одинаковыми наборами весов. Данный вид сетей позволяет сравнить вектора признаков двух объектов с целью выделить их семантическое сходство или различие. Сиамская нейронная сеть представляет собой нелинейное отображение данных с целью приблизить друг к другу схожие объекты и разнести различные объекты на максимально возможное расстояние. Сиамские сети получили свое название от сиамских близнецов, физически приросших друг к другу, из-за использования сразу двух подсетей, разделяющих один набор весов.



* Наши входные данные - это изображения 224x224x3.
* Первый слой свертки использует 96 ядер свертки размером 11x11 с шагом 4. Затем нормализация и объединение в пул применяются на выходе из слоя. После этого уровня сеть разделяется на две идентичные ветви.
* Второй слой свертки использует 256 ядер свертки размером 5x5 каждый, которые затем снова нормализуются по ответам и объединяются в пул.
* Третий, четвертый и пятый слои идентичны, в них 384 ядра размером 3x3 каждое.
* Из-за ограничений памяти здесь, скорее из-за единичного представления 4096 нейронов, делится его на 2 части, по 2048 в каждой. Каждая часть получает данные из обоих элементов.
* Полностью подключенный слой снова повторяется при получении данных из обоих элементов

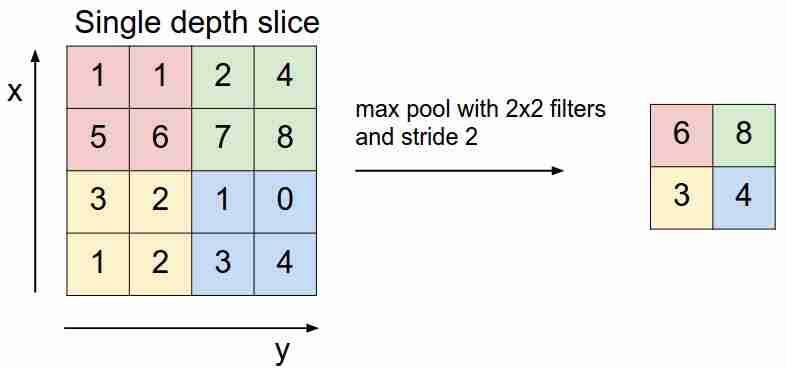
Следующие разделы будут содержать информацию о слоях, используемых в выбранной нами архитектуре. Детали выбора архитектуры и подробности о ней можно будет также прочесть ниже.

# **MaxPool**

В работе использовалась технология MaxPool или Максимальное объединение.

**Максимальное объединение** - это операция, которая используется для уменьшения карты признаков и извлечения наиболее важных функций.

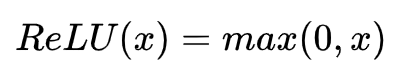
На выходе после слоя max-pooling будет карта признаков, содержащая наиболее заметные признаки предыдущей карты признаков.



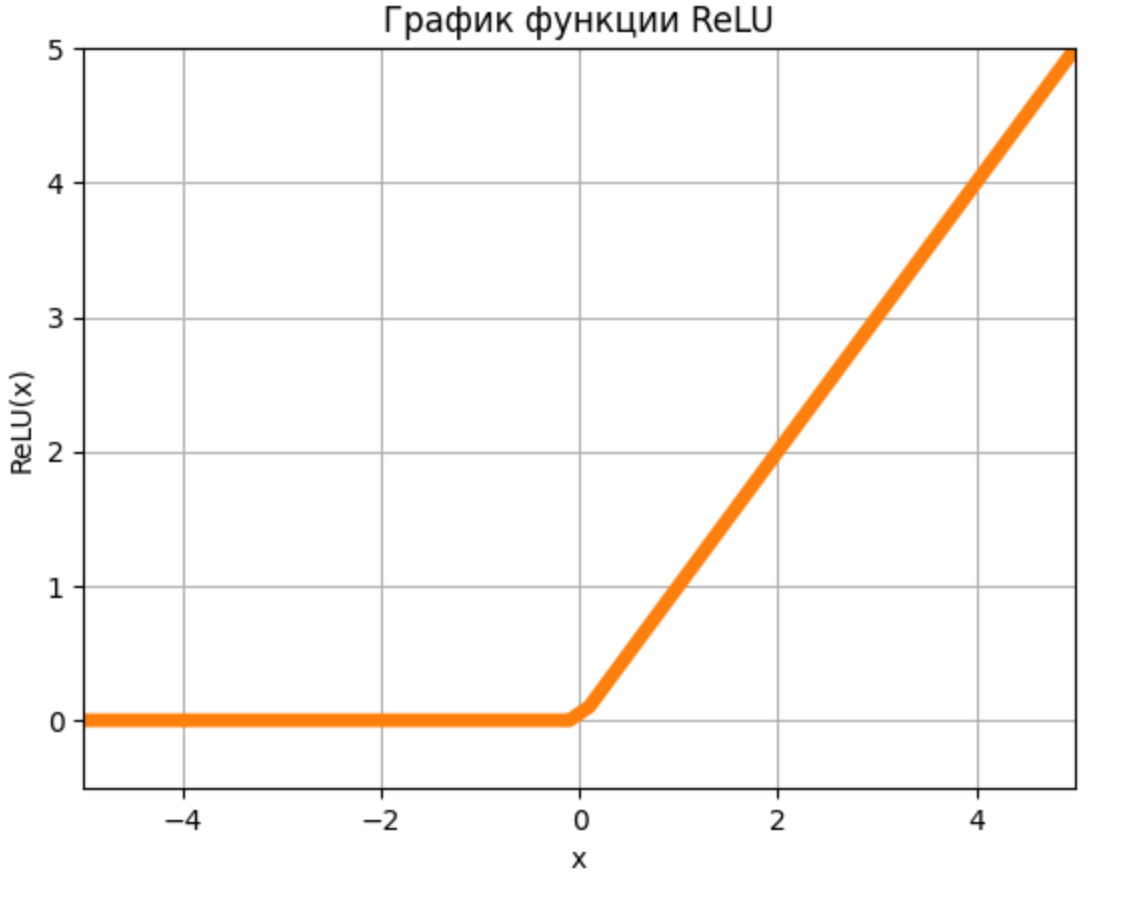
# **Функция активации: ReLU**

ReLU (Rectified Linear Unit) - это нелинейная функция активации[[1]](#footnote-0), которая широко используется в глубоком обучении. Она преобразует входное значение в значение от 0 до положительной бесконечности. Если входное значение меньше или равно нулю, то ReLU выдает ноль, в противном случае - входное значение.

Математически ReLU определяется следующим образом:



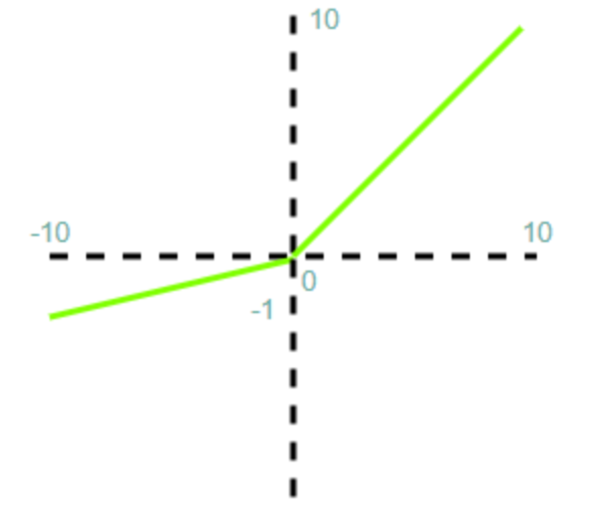
где max - функция, возвращающая максимальное значение из двух.

77

ReLU имеет несколько преимуществ по сравнению со сигмоидной функцией активации. Во-первых, ReLU более вычислительно эффективна, поскольку она является простой и быстрой операцией, которая не требует вычисления экспоненты. Во-вторых, ReLU решает проблему затухания градиента, так как она не вызывает затухания градиента при обратном распространении ошибки, как это происходит в случае с сигмоидной функцией активации.

У данной функции есть один недостаток, называющийся проблемой умирающего ReLU. Так как часть производной функции равна нулю, то и градиент для нее будет нулевым, а это значит, что веса не будут изменяться во время градиентного спуска и этот нейрон, выдавший отрицательное значение, никогда не может быть снова активирован какими-либо данными.

Функция активации Leaky ReLU или LReLU - это другой тип функции активации, которая похожа на ReLU, но решает проблему "отмирания" нейронов, и графически Leaky ReLU обладает следующим преобразующим поведением:



Эта функция очень полезна, поскольку при отрицательных входных данных дифференцирование функции не равно нулю. Следовательно, обучение нейронов не прекращается.

# **Нормализация: batch-normalization**

Рассмотрим классическую нейронную сеть с несколькими слоями. Каждый слой имеет множество входов и множество выходов. Сеть обучается методом обратного распространения ошибки, по батчам[[2]](#footnote-1), то есть ошибка считается по какому-то подмножеству обучающей выборки.

**Пакетная нормализация** (англ. batch-normalization) — метод, который *позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей.* Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

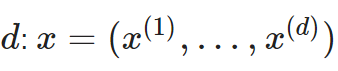
*Пакетный (батч) градиентный спуск* — реализация градиентного спуска, когда на каждой итерации обучающая выборка просматривается целиком, и только после этого изменяются веса модели.Также существует "золотая середина" между стохастическим градиентным спуском и пакетным градиентным спуском — когда просматривается только некоторое подмножество обучающей выборки фиксированного размера. В таком случае такие подмножества принято называть мини-пакетом. Здесь и далее, мини-пакеты будем также называть пакетом.

Пакетная нормализация уменьшает величину, на которую смещаются значения узлов в скрытых слоях (т.н. [ковариантный](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD) сдвиг).

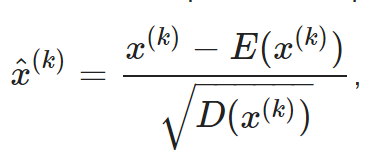
*Ковариантный сдвиг* — это ситуация, когда распределения значений признаков в обучающей и тестовой выборке имеют разные параметры (математическое ожидание, дисперсия и т.д.). Ковариантность в данном случае относится к значениям признаков.

Простой способ решить проблему ковариантного сдвига для входного слоя — это случайным образом перемешать данные перед созданием пакетов. Но для скрытых слоев нейронной сети такой метод не подходит, так как распределение входных данных для каждого узла скрытых слоев изменяется каждый раз, когда происходит обновление параметров в предыдущем слое. Эта проблема называется **внутренним ковариантным сдвигом**. Способом устранения внутреннего ковариантного сдвига является метод пакетной нормализации.

Опишем устройство метода пакетной нормализации. Пусть на вход некоторому слою нейронной сети поступает вектор размерности:

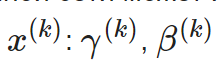


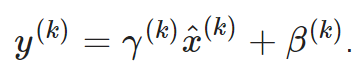
Нормализуем данный вектор по каждой размерности *k:*

,

где математическое ожидание и дисперсия считаются по всей обучающей выборке. Такая нормализация входа слоя нейронной сети может изменить представление данных в слое.

Чтобы избежать данной проблемы, вводятся два параметра сжатия и сдвига нормализованной величины для каждого

 — которые действуют следующим образом:



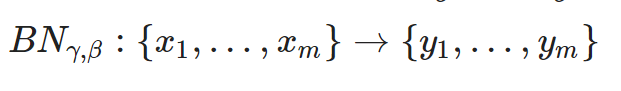
Данные параметры настраиваются в процессе обучения вместе с остальными [параметрами модели](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0_%D0%B8_%D0%B5%D0%B5_%D0%B2%D1%8B%D0%B1%D0%BE%D1%80).

Пусть обучение модели производится с помощью пакетов *B* размера *m*



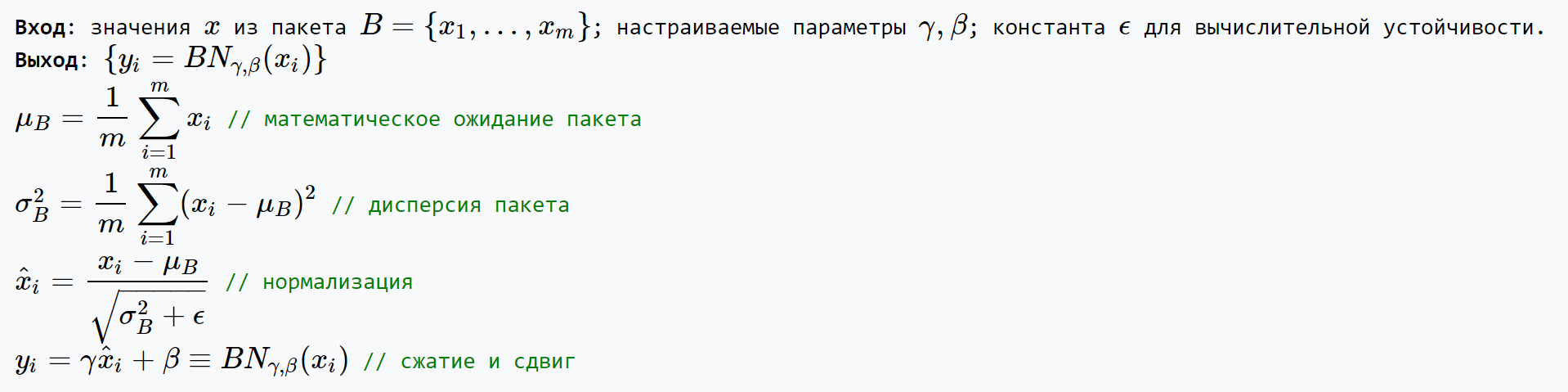
Здесь нормализация применяется к каждому элементу входа с номером k отдельно, поэтому в индекс опускается для ясности изложения. Пусть были получены нормализованные значения пакета  После применения операций сжатия и сдвига были получены 

Обозначим данную функцию пакетной нормализации следующим образом:



Алгоритм пакетной нормализации:

Вход: значения x из пакета B = {x1, …, xm}, настраиваемые параметры ⋎ и β, константа ∈ для вычислительной устойчивости.



# **Регуляризация нейросети: Dropout**

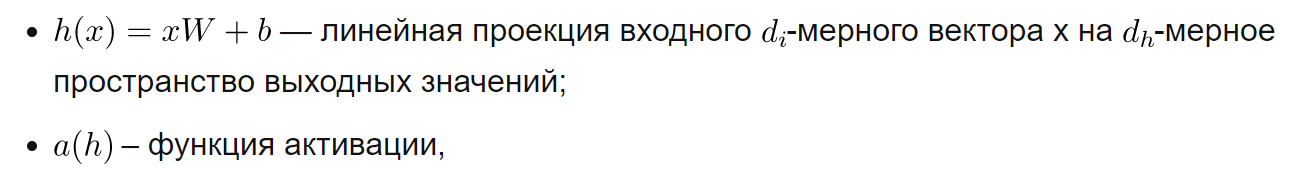
Регуляризация необходима нейронной сети в первую очередь производительности ради, т.к. всегда существует риск переобучения. Переобучение – это такое явление, когда сеть перестает извлекать общеприменимые принципы из обучающего датасета, которые понадобятся для последующей работы, а вместо этого просто учится запоминать особенности полученных входных данных при обучении и ожидаемый для них результат, таким образом симулируя настоящее обучение. В итоге мы получим нейросеть с высокими результатами на обучающем датасете и практически не применимую на других данных.

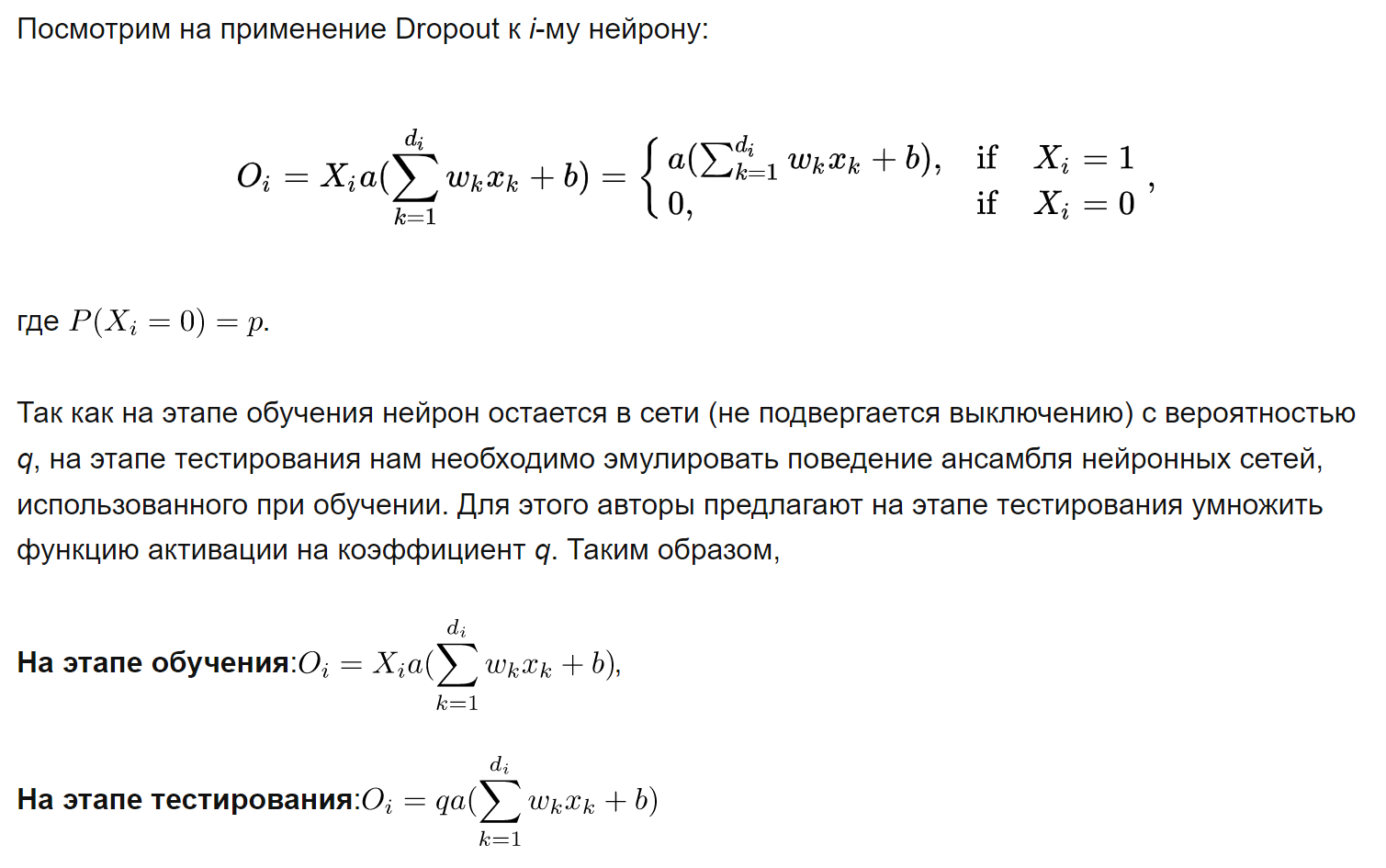
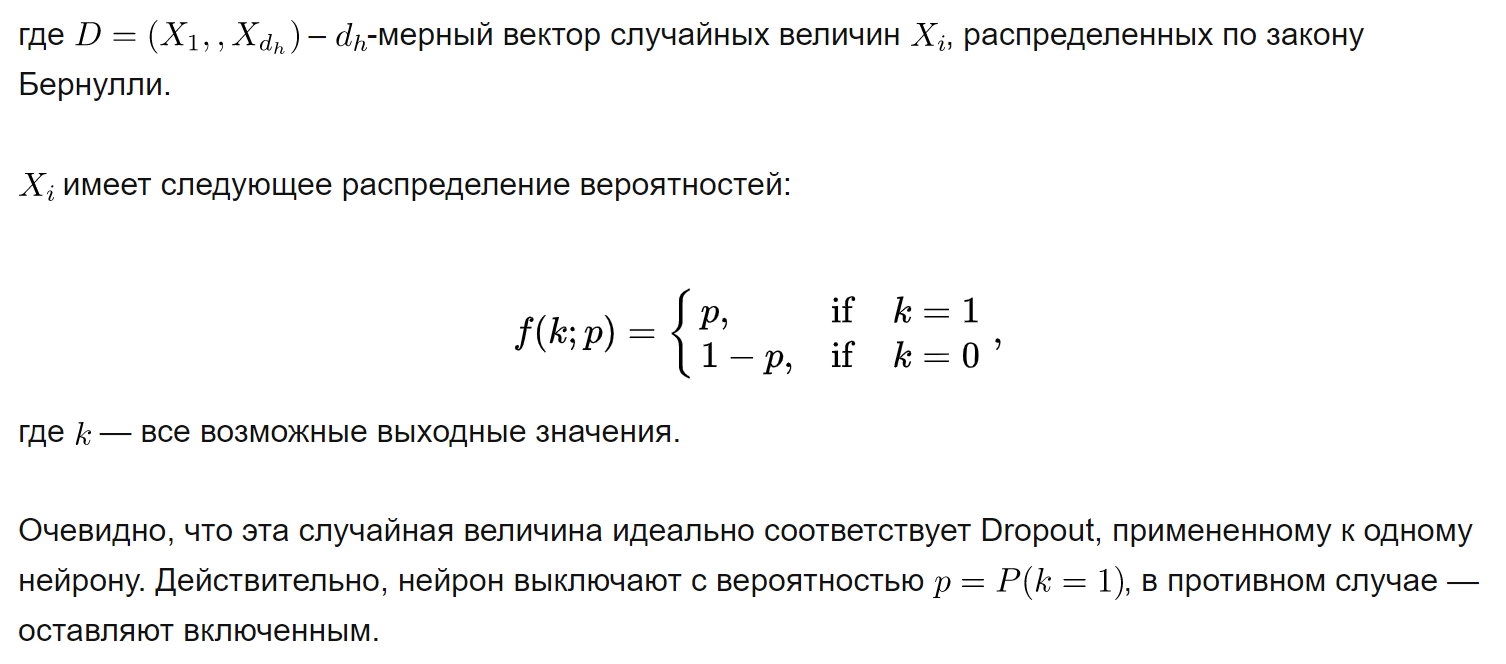
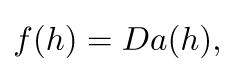
Dropout - это случайное отключение некоторых нейронов, так что каждый нейрон вынужден изучать функции, которые не зависят от его соседей, что приводит к более надежным функциям.

Сети для обучения получаются с помощью исключения из сети (dropping out) нейронов с вероятностью p, таким образом, вероятность того, что нейрон останется в сети, составляет q=1-p. “Исключение” нейрона означает, что при любых входных данных или параметрах он возвращает 0.

Исключенные нейроны не вносят свой вклад в процесс обучения ни на одном из этапов алгоритма обратного распространения ошибки; поэтому исключение хотя бы одного из нейронов равносильно обучению новой нейронной сети.

Вероятность выключения каждого нейрона одинакова. Это означает следующее:

При условии, чтоприменение Dropout к данной проекции на этапе обучения можно представить как измененную функцию активации:

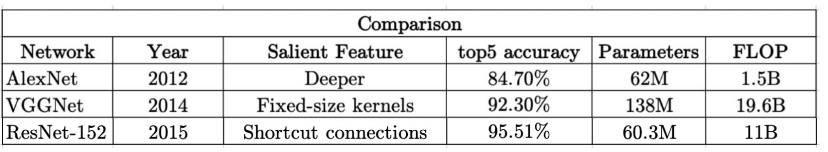


# **Архитектура нейронной сети**

При разработке за основу была взята архитектура сверточной нейронной сети AlexNet,которая оказала большое влияние на развитие машинного обучения, в особенности — на алгоритмы компьютерного зрения. Она довольно популярна в использовании, например, ее применение можно встретить в основе такого нейросетевого трекера как DaSiam RPN.

Этапом прорыва в развитии глубокого обучения стало публикация в 2012 году А.Крижевским, Д.Хинтон и И.Суцкевер архитектуры сверточной нейронной сети под названием «AlexNet», которая с большим успехом победила на конкурсе по распознаванию изображений Image Classification Challenge (ILSVRC). Обучение сети проходило на основе 1,2 миллиона изображений высокого разрешения в 1000 различных классов. При этом, в силу недостаточной мощности графических процессоров обучение производилось на двух графических процессорах с концепцией разделения слоев.

Для сравнения AlexNet с другими архитектурными решениями можно обратиться к таблице ниже, где модели отсортированы с точностью top5 (классификация считается правильной, если любой из пяти прогнозов соответствует целевой метке) на наборе данных IMAGENET. Также можно увидеть количество обучаемых параметров и операций с плавающей запятой (FLOP), необходимых для прямого прохода.



По данным таблицы видно, что:

* AlexNet и ResNet-152 имеют около 60 миллионов параметров, но разница в их точности составляет около 10%. Но обучение ResNet-152 требует большого количества вычислений (примерно в 10 раз больше, чем у AlexNet), а это означает, что требуется больше времени и энергии для обучения.
* VGGNet не только имеет большее количество параметров и FLOP по сравнению с ResNet-152, но и имеет меньшую точность. Обучение сети VGGNet с пониженной точностью занимает больше времени.
* Обучение AlexNet занимает куда меньше времени, чем ResNet-152 и VGGNet, имея меньшии требования к памяти, хоть и уступает им в точности.

Еще одним преимуществом является то, что по сравнению с другими архитектурами в таблице AlexNet имеет относительно простое строение, что хорошо для изучения и самостоятельного создания нейросети. Как правило в ней содержится восемь слоев с весовыми коэффициентами — первые пять из них сверточные, а остальные три — полносвязные. Relu применяется после каждого сверточного и полносвязного слоя. Дропаут применяется перед первым и вторым полносвязными слоями. Слои свертки и максимального объединения чередуются. Подбирается соответствующий задачам алгоритм оптимизации.

**Инструмент реализации: сравнение TF Keras и PyTorch.**

Т.к. в этом семестре мы приступили к самостоятельной реализации нейросети, которая удовлетворяла бы конкретно нашей задаче, встал вопрос в выборе фреймворка для реализации.

За этот семестр мы опробовали и TF Keras, и PyTorch, поэтому можем подвести некоторый итог нашего опыта работы с ними. Стоит упомянуть, что среди наших целей не было подробно сравнить названные библиотеки, потому тут будет приведен лишь наш скромный пользовательский опыт, который, тем не менее, является немаловажным в контексте рассказа о нашей деятельности в этом семестре.

PyTorch мы задействовали и в нашем предыдущем исследовании в рамках альтернативного экзамена прошлого семестра: тогда на нем базировался используемый нами трекер DeepSORT. Правда, тогда преимущественно из библиотеки мы использовали методы, касающиеся работы с элементами данных, например, при использовании в вычислениях тензоров. На этот раз нам предстояло попробовать воспользоваться методами из этого фреймворка для создания моделей.

Для пробы мы выбрали и TF Keras по совету научного руководителя, который уже имел опыт работы с данной библиотекой для написания нейросети.

Сравнивая фреймворки, первую очередь, заметим, что TensorFlow сам по себе старше PyTorch, обладает более широким инструментарием. Keras же можно назвать просто “оберткой” для TF, предоставляющей нам интерфейс для работы с названным фреймворком. Таким образом, при использовании Keras код становится более читабельным и простым для понимания, при этом мы жертвуем скоростью. TF Keras целесообразно использовать для нейронных сетей, чье обучение будет строиться на на небольшом наборе данных – это подходит нам, т.к. мы не планируем делать слишком большой датасет (для относительности “большими” будем считать, например, датасеты, на которых обучаются молодые языковые модели компании OpenAI). Во-первых, это обусловлено отсутствием у нас больших вычислительных мощностей, во-вторых, мы допускаем, что при успешном завершении нынешнего этапа разработки, получения удовлетворяющих нас результатов, мы можем расширить датасет с целью повышения точности распознавания. На данный момент мы разрабатываем прототип, так что Keras в этом плане нам абсолютно подходит.

Тем не менее, мы не отказываемся от возможности использовать в дальнейшем PyTorch, как только прототип будет окончательно готов, т.к. среди преимуществ PyTorch над TF можно выделить простоту отладки. В целом, по многим параметрам они действительно схожи: скорость PyTorch и низкоуровнего API TF почти не различается, а также они оба способны относительно быстро обрабатывать наборы данных большого объема, потому если предстоит выбор, то, пожалуй, сыграет роль именно простота отладки.

# Вывод

По итогам семестра мы изучили, что такое сиамская нейронная сеть, просмотрели готовые решения и сравнили их, в результате чего нами были подробно разобраны детали архитектуры AlexNET и подробности практической части написания собственной нейросети, базирующейся на упомянутой архитектуре. Мы планируем продолжать работать над проектом по завершении семестра в сотрудничестве с Артёмом Юрьевичем, потому все полученные нами знания будут применены на практике. Перспективой данной работы является реализация нейронной сети, соответствующей заявленной задаче и построенной на основе тех выводов, которые мы получили в этом семестре.

Несмотря на то, что на данный момент мы не получили практический результат в силу трудоемкости и объемности задачи, нам кажется важным рассказать и показать ту теорию, которую мы разбирали на протяжении трех месяцев.

1. Функция активации определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения. [↑](#footnote-ref-0)
2. Батч – набор входных выборок, используемых в обучении. [↑](#footnote-ref-1)