**Департамент образования мэрии города Новосибирска**

**Дворец творчества детей и учащейся молодёжи «Юниор»**

**XL городская открытая научно-практическая**

**конференция НОУ «Сибирь»**

Секция «Прикладная информатика и информационные системы»

|  |
| --- |
|  |

**ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

**ТЕМА: «Нейронные сети»**

|  |
| --- |
| Автор:  **Бельков Руслан Марсельевич, 10** класс,  МБОУ АКЛ им. Ю.В. Кондратюка,  Дзержинский район |

Научный руководитель:

**Марковская Людмила Александровна**,

Учитель информатики высшей категории,

Новосибирск, 2021

Оглавление

[Актуальность 2](#_Toc62513158)

[Цели и задачи 3](#_Toc62513159)

[Введение 4](#_Toc62513160)

[История 7](#_Toc62513161)

[Инструментарий 9](#_Toc62513162)

[Структуры и типы нейронных сетей 11](#_Toc62513163)

[Функции активаций 17](#_Toc62513164)

[Обучение 21](#_Toc62513165)

[Практика 25](#_Toc62513166)

[**Лёгкий пример:** 25](#_Toc62513167)

[Первый этап. 25](#_Toc62513168)

[**Второй этап.** 26](#_Toc62513169)

[Третий этап. 26](#_Toc62513170)

[**Сложный пример:** 42](#_Toc62513171)

[Первый этап 43](#_Toc62513172)

[Заключение 50](#_Toc62513173)

[Реквизиты 51](#_Toc62513174)

[Cписок литературы 52](#_Toc62513175)

Актуальность

Машинное обучение – это современный тренд, применяемый в науке и технике. С их помощью улучшаются программы и создаются целые системы, способные автоматизировать, ускорять и помогать в работе человеку. Основная урбанистическая цель – научить систему самостоятельно принимать решения в сложных ситуациях так, как это делает человек.

Разработка НС является одной из самых экономически-перспективных стезей в современном программировании:

**«**Можно взять любую область, в которой используется низкоквалифицированный труд, - например, работу операторов колл-центров - и просто вычесть все людские ресурсы. Я бы сказал, что речь идет о многомиллиардном рынке даже в рамках отдельной страны. Какое количество людей в мире задействовано на низкоквалифицированной работе, можно легко понять. Так что даже очень абстрактно говоря, думаю, речь идет о стомиллиардном рынке во всем мире**»**, — говорит Григорий Бакунов, директор по распространению технологий **«Яндекса».**

**«**Алгоритмы машинного обучения - это следующий шаг в автоматизации любых процессов, в разработке любого программного обеспечения. Поэтому рынок как минимум совпадает со всем рынком ПО, а, скорее, превосходит его, потому что становится возможно делать новые интеллектуальные решения, недоступные старому ПО**»**, — руководитель направления **«Поиск Mail.ru» в Mail.Ru Group** Андрей Калинин.

В конце января 2020 года компания **CB Insights** провела ежегодный анализ глобальных тенденций инвестирования в искусственный интеллект и сообщила, что в 2019 году специализирующиеся на таких технологиях стартапы привлекли рекордные вложения — $26,6 млрд, заключив более 2200 сделок по всему миру.

Цели и задачи

**Целями** моей работы можно назвать изучение принципов работы НС и применение их на практике

**Задачи:**

**1.** Узнать историю становления НС;

**2.** Выбрать инструментарий;

**3.** Изучить структуры и типы НС, примеры использования выбранных инструментов в сфере НС;

**4.** Осознать математические основы НС;

**5.** Применить все полученные знания на простом практическом примере;

**6.** Создать библиотеку, которая должна помочь в будущих проектах, связанных с НС;

Введение

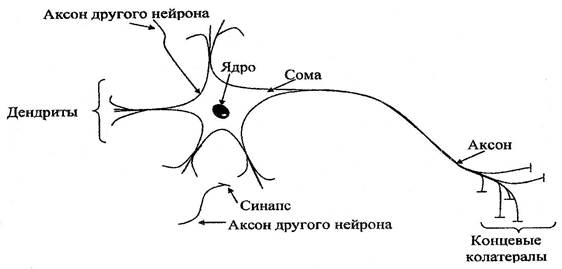
Еще на заре компьютерной эры были намечены два принципиально разных подхода к обработке информации: последовательная обработка символов и параллельное распознавание образов. И символы и образы — это «слова», которые обрабатывают компьютер, а основное различие между ними заключается лишь в размерности. При этом размер образа может быть на много порядков больше размера символа. Казалось бы, разница не очень значительна и приводит лишь к несколько большему времени обработки длинных слов, но на самом деле различия в размерах данных имеют принципиальное значение, так как сложность работы с образами возрастает слишком быстро с изменением их разрядности.

Если для относительно коротких символов можно описать все возможные над ними операции и создать процессор, который предсказуемым образом обрабатывает все входящие символы, исполняющие роль команд или данных, то реализовать то же самое для образов невозможно, поскольку подобное описание будет расти экспоненциально. А значит, любой процессор, предназначенный для обработки образов, содержит лишь часть возможных входных образцов и соответствующих им действий и должен «додумывать» свое поведение и обобщать известные ему примеры, чтобы его реакция была аналогичной и приемлемой с точки зрения решения задачи, для которой он предназначен. Таким образом, различие между последовательными и параллельными вычислениями заключается в принципиально разных методах постановки и решения задач, связанных с обработкой информации.

Для решения задач второго типа необходим алгоритм, который способен обобщать, мыслить почти как человеческий мозг. Биологическая эволюция, которая привела к эффективным решениям обобщения, шла по пути от образов к логике. Так и человек после рождения сначала учится распознавать образы, а только потом приобретает умение рассуждать логически и строить алгоритмы. Компьютеры же, напротив, начав с логики, лишь спустя несколько десятилетий осваивают распознавание образов за счет создания специальных программ для компьютеров или благодаря созданию специализированных аппаратных нейропроцессоров.

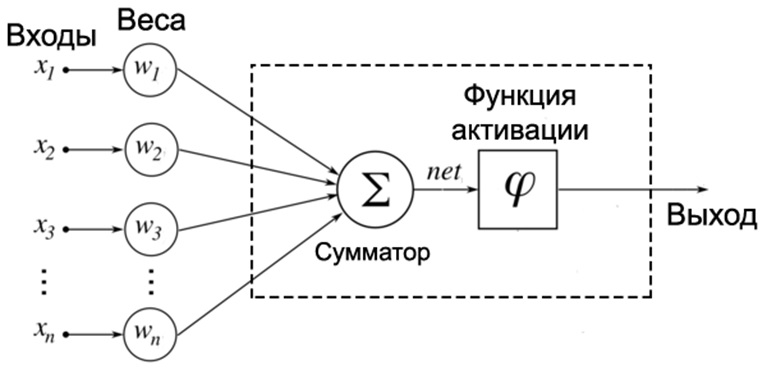
Итак, рассмотрим эволюционное решение:

биологическая нейронная сеть - часть нервной системы, что находится в мозге человека. Совокупность нейронов и сети позволяет нам думать, принимать решения и воспринимать окружающий мир. Биологический нейрон – клетка, основные составляющие которой: ядро, отростки, тела и иные компоненты для тесной связи с тысячами других нейронов. По этой связи передаются электрохимические импульсы, приводящие нейронную сеть в состояние спокойствия или возбуждения.



Нейрон состоит из тела клетки и ядра. К телу относится множество ответвлений, называемых дендритами. Длинные дендриты называться аксонами, с их помощью проходит связь между нейронами, а место контактов двух нейронов называют синапсом. Сигнал приходит на дендрит, передается на другую нейронную клетку по аксону. В точке синаптической передачи импульса, его частота и амплитуда может изменяться. Примерно так работает естественная нейросеть.

В таком случае, создать искусственную нейросеть наподобие естественной не так уж и сложно. Дендрит будет выступать в роли входа, аксон - выхода, тело нейрона в виде сумматора, а кодом(функцией активации и прочим) можно регулировать частоту и импульс сигнала. Это основная составляющая нейрона.

 Графическую модель нейрона можно записать так:

Изучив основные аспекты, можно дать определение искусственной нейронной сети - это модели биологических нейронных сетей мозга, нейроны которых имитируются относительно простыми, часто однотипными, элементами (искусственными нейронами).

Идея нейронных сетей родилась в рамках теории искусственного интеллекта, в результате попыток имитировать способность биологических нервных систем обучаться и исправлять ошибки. Нейронные сети широко используются для решения разнообразных задач. Среди областей применения нейронных сетей: автоматизация процессов распознавания образов, прогнозирование, адаптивное управление, создание экспертных систем, организация ассоциативной памяти, обработка аналоговых и цифровых сигналов, синтез и идентификация электронных цепей и систем. Модели нейронных сетей могут быть программного и аппаратного исполнения.

История

Термин «нейронная сеть» появился в середине XX века. Первые работы, в которых были получены основные результаты в данном направлении, были проделаны Мак-Каллоком и Питтсом. В 1943 году ими была разработана компьютерная модель нейронной сети на основе математических алгоритмов и теории деятельности головного мозга. Они выдвинули предположение, что нейроны можно упрощённо рассматривать как устройства, оперирующие двоичными числами, и назвали эту модель «пороговой логикой».

В 1948 году была опубликована книга Н. Винера о кибернетике. Основной идеей является представление сложных биологических процессов математическими моделями.

В 1949 году канадский физиолог и психолог Дональд Хебб высказал идеи о характере соединения нейронов мозга и их взаимодействии. Он первым предположил, что обучение заключается в первую очередь в изменениях силы синаптических связей. Теория Хебба считается типичным случаем самообучения, при котором испытуемая система спонтанно обучается выполнять поставленную задачу без вмешательства со стороны экспериментатора.

В 1954 году в Массачусетском технологическом институте с использованием компьютеров Фарли и Кларк разработали имитацию сети Хебба. Также исследования нейронных сетей с помощью компьютерного моделирования были проведены Рочестером, Холландом, Хебитом и Дудой в 1956 году.

В 1957 году Розенблаттом были разработаны математическая и компьютерная модели восприятия информации мозгом на основе двухслойной обучающейся нейронной сети. При обучении данная сеть использовала арифметические действия сложения и вычитания. Розенблатт описал также схему не только основного перцептрона, но и схему логического сложения. В 1958 году им была предложена модель электронного устройства, которое должно было имитировать процессы человеческого мышления, а два года спустя была продемонстрирована первая действующая машина, которая могла научиться распознавать некоторые из букв, написанных на карточках, которые подносили к его сенсорам.

Интерес к исследованию нейронных сетей угас после публикации работы по машинному обучению Минского и Пейперта в 1969 году. Ими были обнаружены основные вычислительные проблемы, возникающие при компьютерной реализации искусственных нейронных сетей. Первая проблема состояла в том, что однослойные нейронные сети не могли реализовать функцию «Исключающее ИЛИ». Второй важной проблемой было то, что компьютеры не обладали достаточной вычислительной мощностью, чтобы эффективно обрабатывать огромный объём вычислений, необходимый для больших нейронных сетей.

Исследования нейронных сетей замедлились до того времени, когда компьютеры достигли больших вычислительных мощностей. Одним из важных шагов, стимулировавших дальнейшие исследования, стала разработка в 1975 году Вербосом метода обратного распространения ошибки, который позволил эффективно решать задачу обучения многослойных сетей и решить проблему с функцией

Алгоритм параллельной распределённой обработки данных в середине 1980 годов стал популярен под названием коннективизма. В 1986 году в работе Руммельхарта и Мак-Клелланда коннективизм был использован для компьютерного моделирования нейронных процессов.

Несмотря на большой энтузиазм, вызванный в научном сообществе разработкой метода обратного распространения ошибки, это также породило многочисленные споры о том, может ли такое обучение быть на самом деле реализовано в головном мозге. Отчасти это связывали с тем, что механизм обратного прохождения сигнала не был очевидным в то время, так как не было явного источника обучающего и целевого сигналов. Тем не менее, в 2006 году было предложено несколько неконтролируемых процедур обучения нейронных сетей с одним или несколькими слоями с использованием алгоритмов глубокого обучения. Эти алгоритмы могут быть использованы для изучения промежуточных представлений, как с выходным сигналом, так и без него, чтобы понять основные особенности распределения сенсорных сигналов, поступающих на каждый слой нейронной сети.

Инструментарий

Для реализации примеров необходим удобный и функциональный язык в сфере машинного обучения. Одни из лучших кандидатов для этой роли - **Scala, TypeScript, R, Shell, Julia, C#, Java, JavaScript, C++, Python.**

Я предпочёл **Python** по нескольким причинам:

* широчайший выбор предварительно настроенных инструментов для внедрения МО-моделей и алгоритмов;
* простота синтаксиса, прекрасная читабельность;
* высокая скорость разработки;
* популярность языка среди ML-специалистов;
* динамическая типизация;
* я знаком с данным языком, более близко нежели чем с другими;

Но у него есть минус – относительно Си-подобных и т.д. языков Питон довольно медленный;

Также предполагается возможное будущее использование библиотек **JSON, NumPy, Pandas, Scikit-learn, SciPy, LightGBM, CUDA, Keras, и др.**

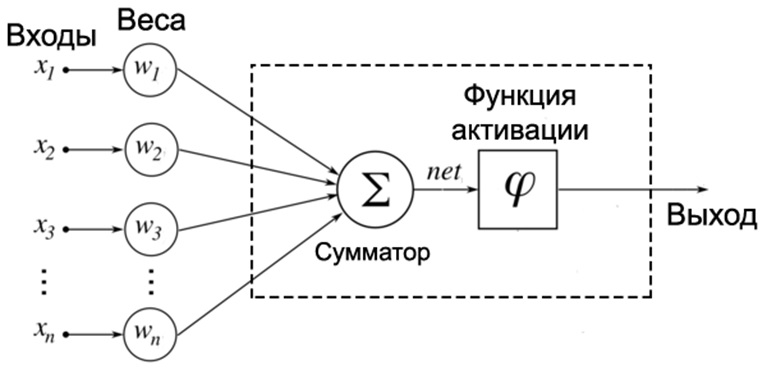
Так как разработка НС может требовать больших объемов ресурсов ПК, нужна довольно мощная машина. Основные характеристики машины, на которой производились все расчеты в этом проекте (в случае если будет недостаточно можно воспользоваться сервисом **Azure M,** также прикольной фичей является **Google Colaboratory**, её можно сразу поставить на гуглдиск и , не устанавливая на компьютер вообще ничего, писать нейронки):

* материнская плата - B85M-G ASUS;
* процессор - Intel Core i5 4460 3.20 GHz;
* видеокарта - GTX 1060 3 Gb;
* оперативная память - 2 x 8 Gb, DDR3, 798.1 MHz;

Структуры и типы нейронных сетей

Нейронная сеть представляет собой совокупность нейронов, которые составляют слои. В каждом слое нейроны между собой никак не связаны, но связаны с нейронами предыдущего и следующего слоев. Информация поступает с первого на второй слой, со второго — на третий и т.д.

Количество слоев и нейронов в них определяют точность и достоверность получаемых результатов при решении задач, т. е. чем больше слоев и нейронов на каждом слое — тем меньше ошибок и выше надежность работы сети. Однако, если построить слишком большую сеть, то можно столкнуться с уменьшением производительности и увеличением сложности модели. Потому при выборе архитектуры сети следует принимать во внимание условия решаемой задачи. Так как нейронные сети строятся на искусственных нейронах, имеет смысл рассмотреть их строение и функционирование.

 Главная функция искусственного нейрона — формировать выходной сигнал в зависимости от сигналов, поступающих на его входы. В самой распространенной конфигурации входные сигналы обрабатываются адаптивным сумматором, который выполняет сложение сигналов, поступающих по синаптическим связям от других. Затем выходной сигнал сумматора поступает в нелинейный преобразователь для вычисления состояния нейрона, где преобразуется функцией активации, и результат подается на выход. Нейрон характеризуется текущим состоянием и обладает группой синапсов — однонаправленных входных связей, соединенных с выходами других нейронов. Общий вид искусственного нейрона, который уже приводился:

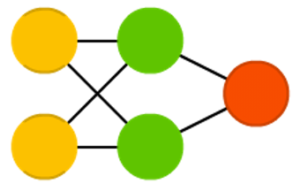
Нейрон имеет аксон — выходную связь данного нейрона, с которой сигнал поступает на синапсы следующих нейронов. Каждый синапс характеризуется величиной синаптической связи (ее весом Wi). Текущее состояние нейрона определяется как взвешенная сумма его входов(S)

Выход нейрона есть функция его состояния: ,

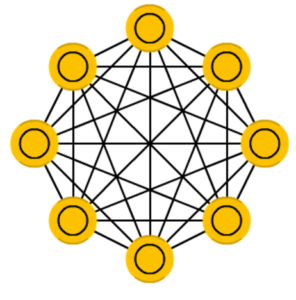
где **Yp** - выход формального нейрона P, **f** - активационная функция, **Wi** - вес связи, входящий в нейрон P, **Yi** - выход нейрона, из которого эта связь исходит.

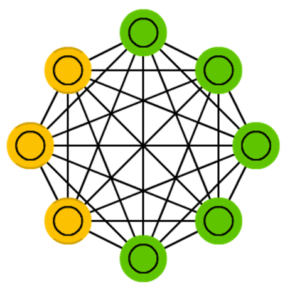
Активационная функция, вычисляющая выходной сигнал нейрона.

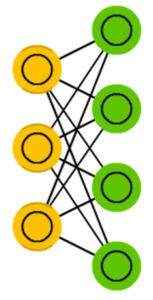
Сегодня существует огромное количество различных конфигураций нейронных сетей с различными принципами функционирования:

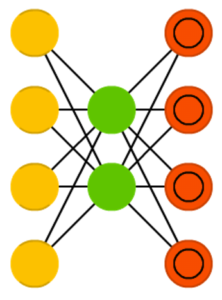
 Нейронные **сети прямого распространения** (**FFNN**) и **перцептроны** (P) очень прямолинейны, они передают информацию от входа к выходу. Прямое распространение сигнала означает, что такая нейронная сеть не содержит петель;

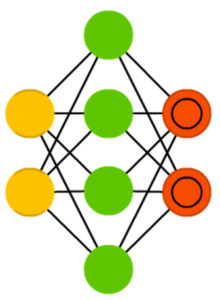
**Сети радиально-базисных функций** (**RBF**) — это **FFNN**, которая использует радиальные базисные функции как функции активации. Больше она ничем не выделяется.

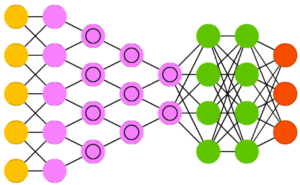
 **Нейронная сеть Хопфилда** (**HN**) — это полносвязная нейронная сеть с симметричной матрицей связей. Во время получения входных данных каждый узел является входом, в процессе обучения он становится скрытым, а затем становится выходом.

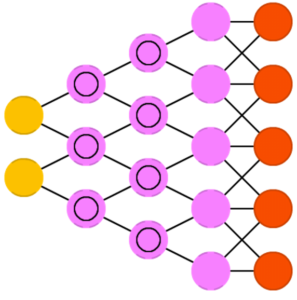
** Машина Больцмана** (**BM**) очень похожа на сеть Хопфилда, но в ней некоторые нейроны помечены как входные, а некоторые —как скрытые. Входные нейроны в дальнейшем становятся выходными. Это стохастическая сеть.

** Ограниченная машина Больцмана** (**RBM**). Единственной разницей с **BM** является её ограниченность. В ней нейроны одного типа не связаны между собой.

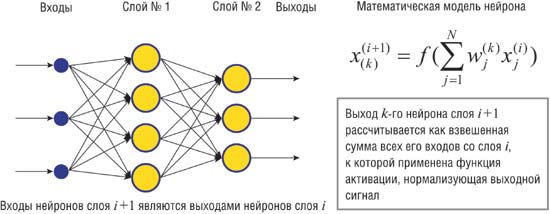
 **Автокодировщик** (**AE**). Это другой способ использования FFNN. Основной идеей является автоматическое сжатие информации. Сама сеть по форме напоминает песочные часы, в ней скрытые слои меньше входного и выходного, она симметрична.

** Разреженный автокодировщик** (**SAE**) — противоположность обычного. Вместо того, чтобы обучать отображать информацию в меньшем количестве узлов, мы увеличиваем их количество. Сеть к центру раздувается. Сети такого типа полезны для работы с большим количеством мелких свойств набора данных.

 **Свёрточные нейронные сети** (**CNN**) сильно отличаются от других видов сетей. Обычно они используются для обработки изображений, реже для аудио. Входные данные преобразуются перед входом, разбиваясь на части (можно сказать что к каждой части применяется другая НС). Эти входные данные затем передаются через свёрточные слои, в которых не все узлы соединены между собой. Эти слои имеют свойство сжиматься с глубиной, причём часто используются степени двойки: 32, 16, 8, 4, 2, 1. На практике к концу CNN прикрепляют FFNN для дальнейшей обработки данных. Такие сети называются глубинными (**DCNN**).

 **Развёртывающие нейронные сети** (**DN**), также называемые обратными графическими сетями, являются обратным к свёрточным нейронным сетям. Что на выходе в сверточных подается на вход к разверточным и получается то, что примерно было на входе у сверточной.

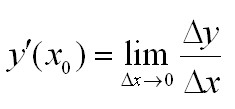
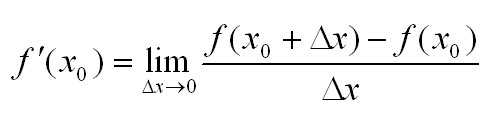
В качестве примера рассмотрим **многослойную полносвязанную нейронную сеть прямого распространения**, которая широко используется для поиска закономерностей и классификации образов. Полносвязанной нейронной сетью называется многослойная структура, в которой каждый нейрон произвольного слоя связан со всеми нейронами предыдущего слоя, а в случае первого слоя — со всеми входами нейронной сети. С прямым распространением мы уже разобрались.



Функции активаций

Активационная (характеристическая) функция — это любая нелинейная функция, вычисляющая выходной сигнал нейрона. Выбор функции определяется спецификой поставленной задачи, либо ограничениями, накладываемыми некоторыми алгоритмами обучения.

В обучении(корректировки весов) нам будет необходима производная от активационной функции. Введем понятие производной:

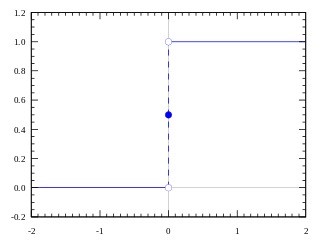
 **Производная функции в точке** – предел отношения приращения функции в данной точке к приращению аргумента, когда последнее стремится к нулю.

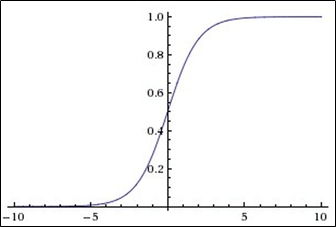
**Геометрический смысл производной**: производная от функции в точке равна тангенсу угла между осью OX и касательной к графику функции в данной точке.

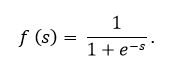


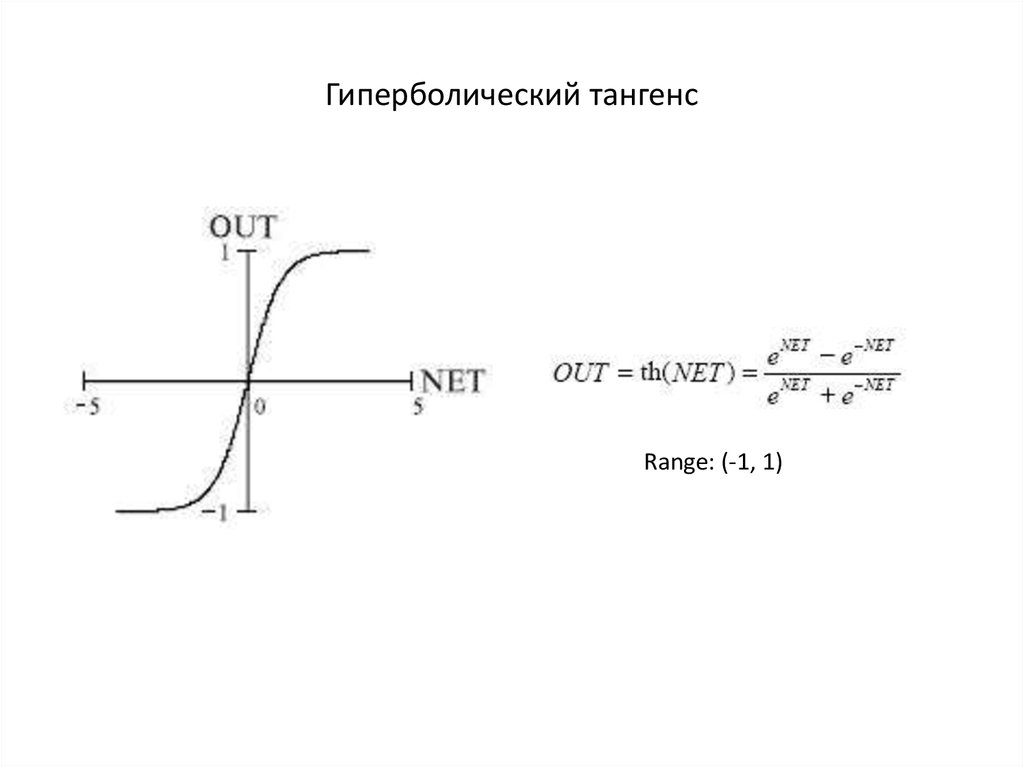
Одним из этапов разработки нейронной сети является выбор функции активации нейронов. Вид функции активации во многом определяет функциональные возможности нейронной сети и метод обучения этой сети.

Рассмотрим наиболее распространенные функций активации, применяемые в нейронных сетях:

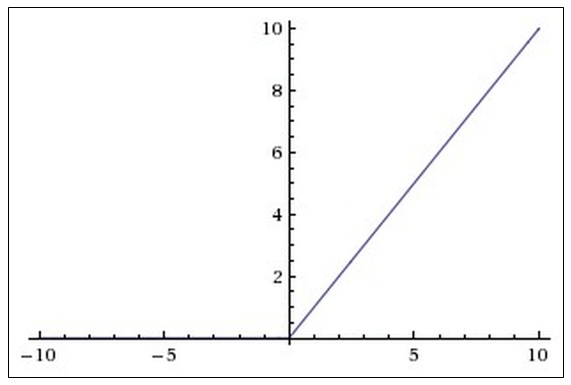
** Ступенчатая функция активации.** НС с такой функцией активации не сможет обобщать.

** Сигмоида.** На выходе дает вещественное число в интервале от 0 до 1. В частности, большие (по модулю) отрицательные числа превращаются в ноль, а большие положительные – в единицу. Исторически сигмоида находила широкое применение, поскольку ее выход хорошо. интерпретируется. Является непрерывной; монотонно возрастающей; дифференцируемой.



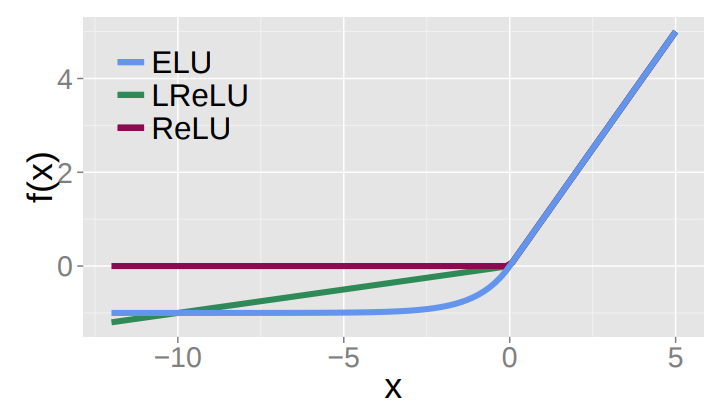
 **Гиперболический тангенс.** Симметричные активационные функции, типа гиперболического тангенса обеспечивают более быструю сходимость, чем стандартная логистическая функция; функция имеет простую производную, которая может быть вычислена через ее значение, что дает экономию вычислений.

**ReLU.** Преимущества использования: ее производная равна либо единице, либо нулю. Вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как ReLU может быть реализован с помощью простого порогового преобразования матрицы активаций в нуле; отсекает ненужные детали в канале при отрицательном выходе. Но есть недостаток - ReLU может выводить нейроны из обучения (об этом можно прочитать в конце главы).



**LReLU(Leaky).** Недостаток обычной ReLU: она может выходить из строя («умирать»). Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Эта проблема решается посредством выбора надлежащей скорости обучения. ReLU с «утечкой» представляет собой одну из попыток решить описанную выше проблему. Обычный ReLU на интервале x < 0 дает на выходе ноль, в то время как LReLU имеет на этом интервале небольшое отрицательное значение (угловой коэффициент около 0,01). То есть функция для LReLU имеет вид f(x) = αx при x < 0 и f(x) = x при x ≥ 0, где α – малая константа. Некоторые исследователи сообщают об успешном применении данной функции активации, но результаты не всегда стабильны.

Производная равна либо 1, либо **a.**



Обучение

Теперь давайте чуть более подробно рассмотрим вопрос обучения нейронной сети. Что это такое? И каким образом это происходит?

**Обучение нейронной сети** (Training) — поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется в нужный нам выходной.

Однако есть еще один важный момент. Если обучать сеть, используя только один входной сигнал, то сеть просто «запомнит правильный ответ». Со стороны будет казаться, что она очень быстро «обучилась». И как только вы подадите немного измененный сигнал, ожидая увидеть правильный ответ, то сеть выдаст бессмыслицу. Зачем нам сеть, определяющая лицо только на одном фото. Мы ждем от сети способности обобщать какие-то признаки и узнавать лица и на других фотографиях тоже. Именно с этой целью и создаются обучающие выборки.

**Обучающая выборка** (Training set) — конечный набор входных сигналов (иногда вместе с правильными выходными сигналами), по которым происходит обучение сети.

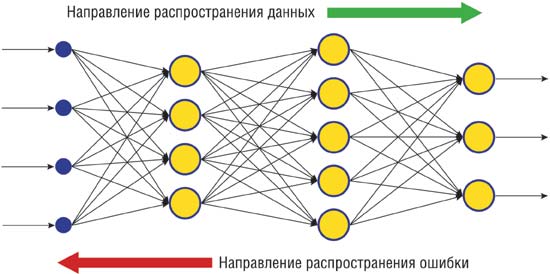
После обучения сети, то есть когда сеть выдает корректные результаты для всех входных сигналов из обучающей выборки, ее можно использовать на практике. Однако прежде чем пускать свежеиспеченную нейросеть "в бой", часто производят оценку качества ее работы на так называемой тестовой выборке.

**Тестовая выборка** (Testing set) — конечный набор входных сигналов, по которым происходит оценка качества работы сети.

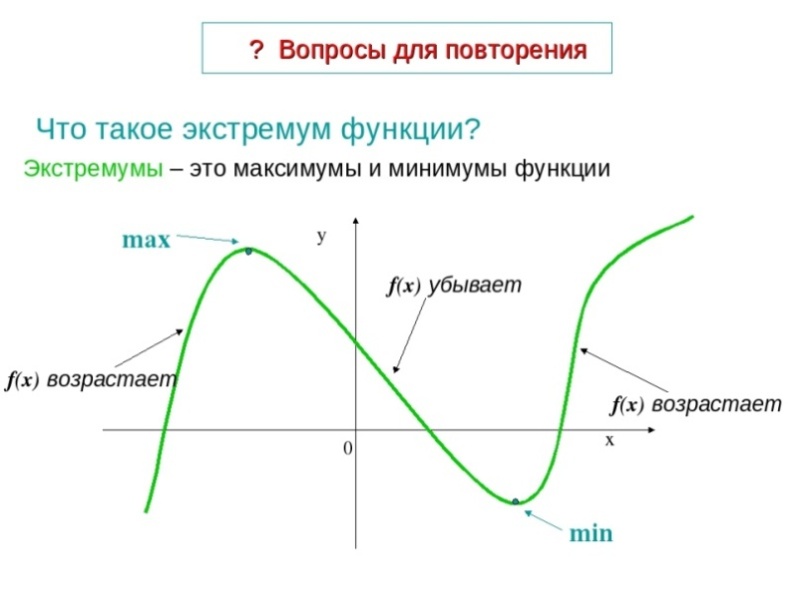
**Обучение с учителем.** Суть данного подхода заключается в том, что вы даете на вход сигнал, смотрите на ответ сета и сравниваете его с уже готовым, правильным ответом. Затем, с помощью специальных алгоритмов, вы меняете веса связей нейронной сети и снова даете ей входной сигнал. Сравниваете ее ответ с правильным и повторяете этот процесс до тех пор, пока сеть не начнет отвечать с приемлемой точностью (100% быть не может).

**Обучение без учителя** применяют тогда, когда у нас нет правильных ответов на входные сигналы. В этом случае вся обучающая выборка состоит из набора входных сигналов. При таком «обучении» сеть начинает выделять классы подаваемых на вход сигналов. Сеть начинает проводить кластеризацию. Например, вы демонстрируете сети конфеты, пирожные и торты. Вы никак не регулируете работу сети. Вы просто подаете на ее входы данные о данном объекте. Со временем сеть начнет выдавать сигналы трех разных типов, которые и отвечают за объекты на входе.

Существует большое число алгоритмов обучения, ориентированных на решение разных задач. Среди них выделяет алгоритм обратного распространения ошибки, который является одним из наиболее успешных современных алгоритмов. Его основная идея заключается в том, что изменение весов синапсов происходит с учетом локального градиента функции ошибки. Разница между реальными и правильными ответами нейронной сети, определяемыми на выходном слое, распространяется в обратном направлении — навстречу потоку сигналов. В итоге каждый нейрон способен определить вклад каждого своего веса в суммарную ошибку сети. Простейшее правило обучения соответствует методу наискорейшего спуска, то есть изменения синаптических весов пропорционально их вкладу в общую ошибку.

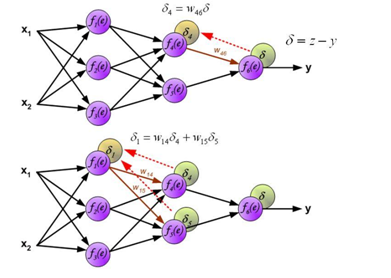


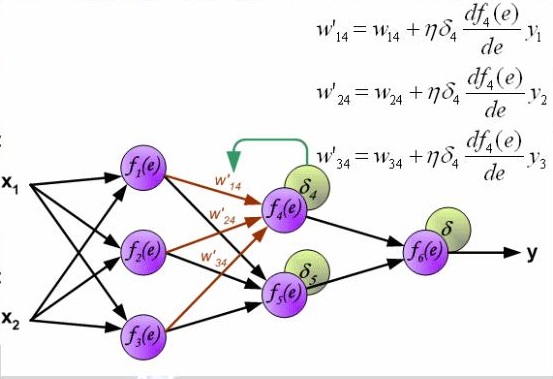
При таком обучении нейронной сети нет уверенности, что она обучилась наилучшим образом, поскольку всегда существует возможность попадания алгоритма в локальный минимум. Для этого используются специальные приемы, позволяющие «выбить» найденное решение из локального экстремума. Если после нескольких таких действий нейронная сеть сходится к тому же решению, то можно сделать вывод о том, что найденное решение, скорее всего, оптимально.

(Для функции ошибки на сете.)

Классический **алгоритм обратного распространения ошибки** хорошо работает на двухслойных и трехслойных нейронных сетях, но при дальнейшем увеличении глубины начинает испытывать проблемы. Одна из причин — так называемое затухание градиентов. По мере распространения ошибки от выходного слоя к входному на каждом слое происходит домножение текущего результата на производную функции активации. Производная у традиционной сигмоидной функции активации меньше единицы на всей области определения, поэтому после нескольких слоев ошибка станет близкой к нулю. Если же, наоборот, функция активации имеет неограниченную производную (как, например, гиперболический тангенс), то может произойти взрывное увеличение ошибки по мере распространения, что приведет к неустойчивости процедуры обучения.

Стандартный метод обратного распространения ошибки:

 Нейронная сеть инициализируется по обучающей выборке. После полного прохождения сети мы получаем некий результат(назовем его реальным). Для ошибки последнего нейрона всё просто - вычитаем из желаемого реальный результат. Для нейрона скрытого слоя - ошибка равна сумме произведений всех ошибок нейронов, связанных с ним, следующими по вектору распространения данных, на соответствующие веса.

Сама **корректировка** весов происходит так: 

То есть, корректировочный вес нейрона равен = (весу нейрона) + (коэффициент обучения) \* (ошибку нейрона, В который входит эта связь) \* (производную функции активации, по последнему аргументу, который входил В этот нейрон) \* (последний выход нейрона, ИЗ которого исходит связь).

Практика

## **Лёгкий пример:**

Ну а теперь, переходим от теории к практике. Я воспользуюсь прекрасными библиотеками от tensorflow и ноутбуком Jupyter. Программировать все методы НС не имеет смысла, т.к. готовые решения зачастую быстрее(во многом из-за того, что написаны на более быстрых языках). Итак, задача – распознавание собак и кошек. План:

1. Создать виртуальную среду разработки с tensorflow(нужно установить CUDA Toolkit и cudnn, tensorflow-database и настроить совместимость под подходящий cuda драйвер);
2. Поставить и запустить Jupyter Notebook, проверить работоспособность виртуальной среды, видимость видеокарты;
3. Скачать необходимые базу данных для обучения, модель; Обучить модель и сохранить значение весов;
4. Вроде бы должно работать…
5. Фиксим баги;
6. Работает!

### Первый этап.

Быстрее и проще всего это сделать при помощи Anaconda, т.к. она поддерживает контроль версий, а также не нужно получать профиль разработчика на оф. сайте NVIDIA.

1. Качаем её с оф. сайта (<https://www.anaconda.com>);
2. Ставим;
3. Релогаем комп, если попросят;
4. Запускаем Anaconda Navigator;
5. Авторизуемся(регистрируемся), если просит;
6. Открываем вкладку Environments;
7. Настраиваем среду:
   1. Нажимаем на стрелочку около нашей среды;
   2. Open Terminal
   3. conda install -c conda-forge tensorflow
   4. pip freeze | findstr tensorflow
   5. Получаем версию модуля, заходим на <https://www.tensorflow.org/install/source>, доходим до Tested build configurations, ищем GPU сборку под вашу версию tensorflow; запоминаем версии CUDA и cuDNN.
   6. conda install -c anaconda cudatoolkit=подходящая версия
   7. conda install -c anaconda cudnn=подходящая версия
   8. conda install -c anaconda tensorflow-datasets
8. Первый этап должен быть закончен.

**Второй этап.**

1. В Anaconda Navigator заходим во вкладку Home;
2. Находим Jupyter Notebook и скачиваем();
3. Заходим в директорию, в который хотим делать проект, с шифтом нажимаем на пустое место внутри папки – открыть окно PowerShell…
4. jupyter notebook
5. Справа ищем NEW 🡪 Python X 🡪 Открывается новый блокнот;
6. В первой ячейке пишем:

**import** numpy **as** np

**import** tensorflow **as** tf

**import** tensorflow\_datasets **as** tfds

**from** tensorflow.keras.preprocessing.image **import** load\_img, img\_to\_array

**from** tensorflow.keras.layers **import** Dense, GlobalAveragePooling2D, Dropout

**import** matplotlib.pyplot **as** plt

1. Во второй ячейке пишем:

**import** os

os.environ["CUDA\_DEVICE\_ORDER"] = "PCI\_BUS\_ID"

os.environ["CUDA\_VISIBLE\_DEVICES"] = "0, 1"

os.environ['TF\_FORCE\_GPU\_ALLOW\_GROWTH'] = 'true'

**import** tensorflow **as** tf

gpus = tf.config.experimental.list\_physical\_devices('GPU')

**if** gpus:

*# Restrict TensorFlow to only allocate 2GB of memory on the first GPU*

**try**:

    logical\_gpus = tf.config.experimental.list\_logical\_devices('GPU')

**print**(len(gpus), "Physical GPUs,", len(logical\_gpus), "Logical GPUs")

**except** RuntimeError **as** e:

*# Virtual devices must be set before GPUs have been initialized*

**print**(e)

Запускаем, у второго блока должно быть в выводе что-то около: 1 Physical GPUs, 1 Logical GPUs.

1. Если всё в порядке, этап закончен, иначе – фиксим баги; Тут могу только удачи пожелать…

### Третий этап.

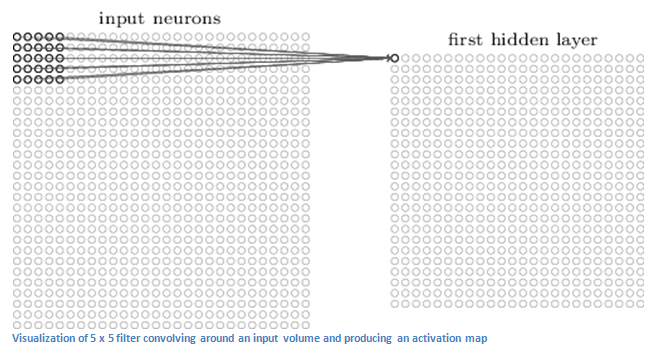
1. train\_data, metadata = tfds.load('cats\_vs\_dogs', split=['train'], with\_info=1, as\_supervised=1) Это всё в терминале, грузиться будет довольно долго
2. Вот сам код, через *enter* обозначены cells.

size = 224 # определяем размер картинки, который подходит под модель(224 именно под ту, которую скачаем)   
  
train\_data, metadata = tfds.load('cats\_vs\_dogs', split=['train'], with\_info=1, as\_supervised=1)  
  
  
# загружаем бд для обучения, метадата - данные об самой бдшке, они нам не понадобятся  
  
# функцией приводим все картинки к подходящему размеру(224х224)  
def resize\_image(img, label):  
 img = tf.cast(img, tf.float32)  
 img = tf.image.resize(img, (size, size))  
 img = img / 255  
 return img, label  
  
train\_data\_resized = train\_data[0].map(resize\_image)  
train\_batches = train\_data\_resized.shuffle(500).batch(16)  
  
# загружаем красиво нарисованную модель(сверточная), которая уже есть в tf.  
base\_layers = tf.keras.applications.MobileNetV2(input\_shape=(size, size, 3), include\_top=0)  
base\_layers.trainable = 0  
  
# вот так мы задаем экземпляр класса модели  
model = tf.keras.Sequential([  
  
 base\_layers,  
 GlobalAveragePooling2D(),  
 Dropout(0.1),  
 Dense(1)  
])  
model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(from\_logits=True), metrics=['accuracy'])  
  
# загружаем значения весов(впервые нужно будет пропускать этот пункт, запуская сразу следующий Cell)  
checkpoint\_path = "training\_1/cp.ckpt"  
model.load\_weights(checkpoint\_path)  
  
# checkpoint\_path = "training\_1/cp.ckpt" - нужно будет раскомментировать впервые  
checkpoint\_dir = os.path.dirname(checkpoint\_path)  
  
# Создаем коллбек сохраняющий веса модели  
cp\_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint\_path,  
 save\_weights\_only=True, verbose=1)  
# Обучаем модель с новым коллбеком  
model.fit(train\_batches,  
 epochs=5,  
 callbacks=[cp\_callback]) # Pass callback to training

# уже обученную сеточку итерируем по папке с котятами и собаками  
for i in range(10):  
 try:  
 img = load\_img('photos/' + f'{i + 1}.jpg')  
 img\_array = img\_to\_array(img)  
 img\_resized, \_ = resize\_image(img\_array, \_)  
 img\_expended = np.expand\_dims(img\_resized, axis=0)  
 prediction = model.predict(img\_expended)[0][0]  
 pred\_label = 'КОТ' if prediction < 0.5 else 'СОБАКА'  
 plt.figure()  
 plt.imshow(img)  
 plt.title(f'{pred\_label} {prediction}')  
 except:  
 print('Такого файла нет в директории')

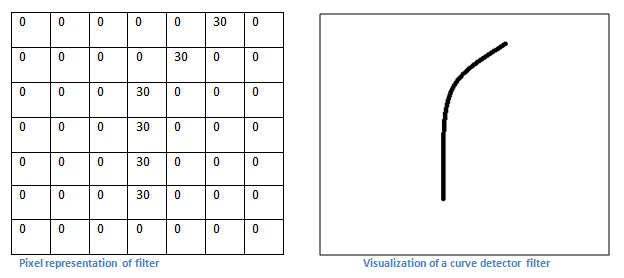
А теперь разберемся, что же за модель мы создали. Во-первых, НС, которые распознают объекты(которые способны классифицировать по изображениям), в основном, являются свёрточными. Используемая в данном коде НС не является исключением, интересность такого типа заключается в том, что на её вход подается изображение по частям.

Первый слой в СНС всегда свёрточный. В нашем случае вводное изображение — это матрица 32х32х3 (RGB) с пиксельными значениями. Легче всего понять, что такое свёрточный слой, если представить его в виде фонарика, который светит на часть изображения. Допустим свет, который излучает этот фонарик, покрывает площадь 5 х 5 и использует только один канал. А теперь давайте представим, что фонарик движется по всем областям вводного изображения. В терминах компьютерного обучения этот фонарик называется фильтром (иногда нейроном или ядром), а области, на которые он светит, называются полем восприятия. То есть наш фильтр — это матрица (такую матрицу ещё называют матрицей весов или матрицей параметров). Теперь давайте за пример возьмем позицию, в которой находится фильтр. Пусть это будет левый верхний угол.

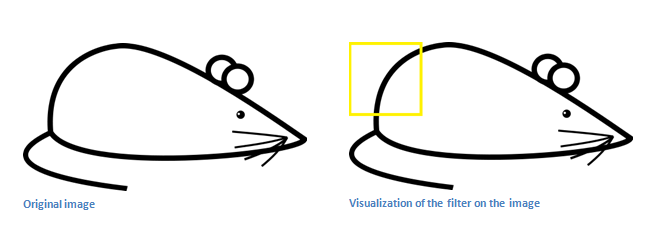


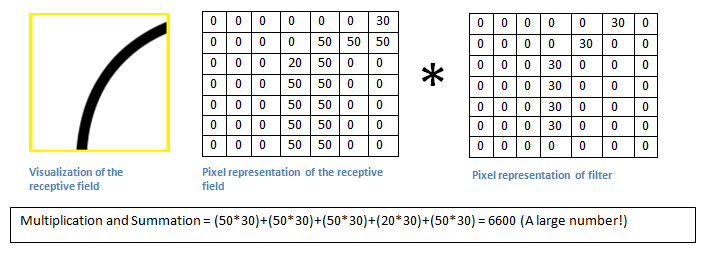
Поскольку фильтр производит свёртку, то есть передвигается по вводному изображению, он умножает значения фильтра на исходные значения пикселей изображения (поэлементное умножение). Все эти умножения суммируются (всего 75 умножений). И в итоге получается одно число. Помните, оно просто символизирует нахождение фильтра в верхнем левом углу изображения. Теперь повторим этот процесс в каждой позиции. (Следующий шаг — перемещение фильтра вправо на единицу, затем еще на единицу вправо и так далее). Каждая уникальная позиция введённого изображения производит число. После прохождения фильтра по всем позициям получается матрица с глубиной 1, которую называют функцией активации или картой признаков.

Каждый фильтр можно рассматривать как идентификатор свойства: прямые границы, простые цвета и кривые. Подумайте о самых простых характеристиках, которые имеют все изображения в общем. Скажем, наш первый фильтр 7 х 7 х 3, и он будет детектором кривых. У фильтра пиксельная структура, в которой численные значения выше вдоль области, определяющей форму кривой (помните, фильтры, о которых мы говорим, это просто числа). Вот сам фильтр:

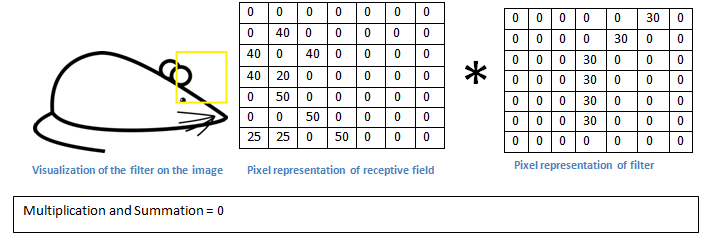


Когда у нас в левом верхнем углу вводного изображения есть фильтр, он производит умножение значений фильтра на значения пикселей этой области. Давайте рассмотрим пример изображения, которому мы хотим присвоить класс, и установим фильтр в верхнем левом углу.





Если на вводном изображении есть форма, в общих чертах похожая на кривую, которую представляет этот фильтр, и все умноженные значения суммируются, то результатом будет большое значение(6600)! Теперь давайте посмотрим, что произойдёт, когда мы переместим фильтр.

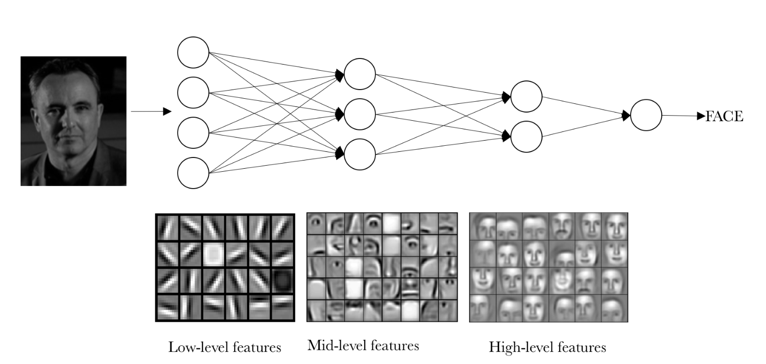


Значение равно нулю, и это неудивительно, в той области действительно нет похожих кривых. В итоге на нейрон первого скрытого слоя будет подано значение 0, а в первом случае – 6600. Так, сеть может разбить всё изображение на простые признаки низкого порядка(прямые, кривые), потом брать несколько таких признаков и определять по ним признаки более высокого порядка(например, геометрические фигуры) и так далее, пока не определится сам необходимый объект.

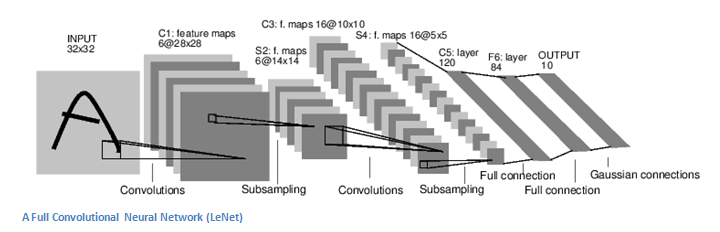
И такого рода слои в СНС архитектуре будут перемежатся с другими. Классическая архитектура СНС выглядит так:



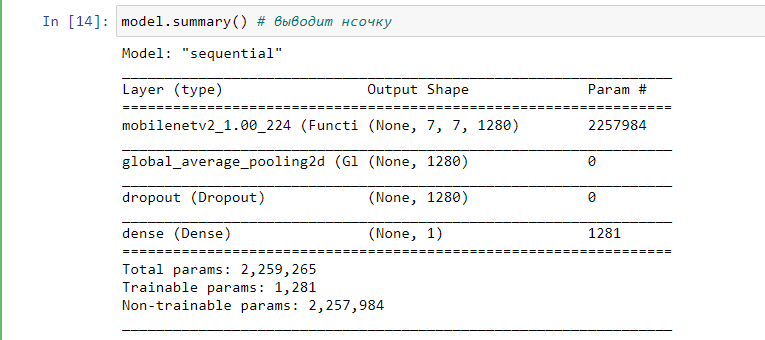
Начальные слои СНС действуют у всех одинаково, т.к. любое изображение можно разбить на низкоуровневые признаки. Поэтому для своих целей обучают только последние слои – полносвязные, а базовые берут из готовых СНС.



Как-то так



Наша сеть выглядит вот-так:



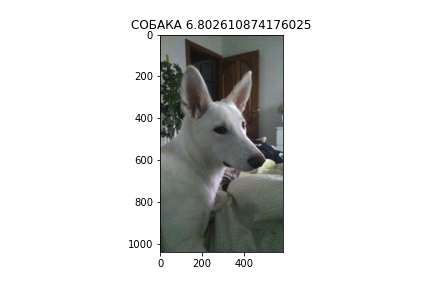
Mobilenetv2 – как раз-таки загруженные базовые слои, можно посмотреть и их строение(base\_layers.summary()), но оно нам не надо…

Dropout – нейроны, которые выпадают из сети, существуют для того, чтобы НС не смогла просто заучить их. Меняются во время обучения.

Dense – чувствительный и обучаемый слой, в нашем случае он один, поэтому его выход – и есть ответ нашей сети.

Global… - «технология кодирования изображений со снижением цветового разрешения, при которой частота выборки цветоразностных сигналов может быть меньше частоты выборки яркостного сигнала» для 2d матриц.

**Далее пойдут тесты**







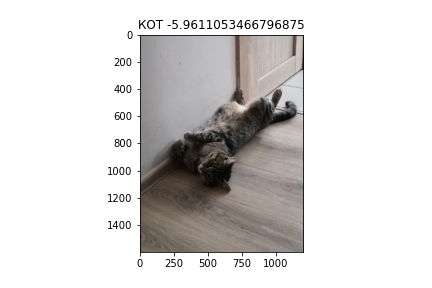




Вообще это кролик, но нейронка считает, что это больше «собака», че «кот». У неё нет никаких альтернатив, либо красная, либо синяя таблетка



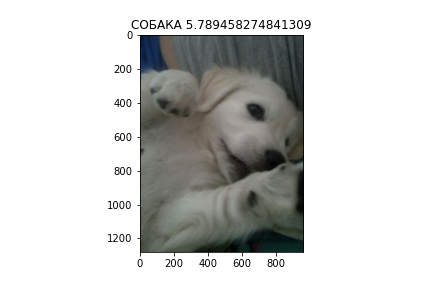


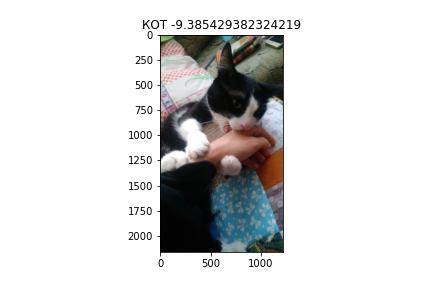


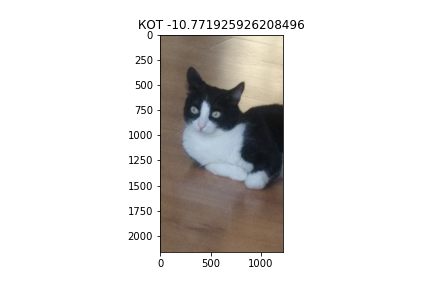




Вообще, это собака, но из-за ракурса и цветовой гаммы НСке сложно определить кто это. Впрочем, мне тоже было сложно определить что это… Также распознаванию очень сильно мешает шум, фильтры и т.д.













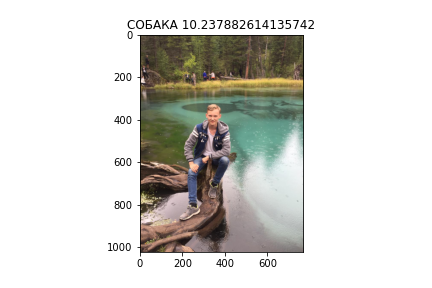




Все числа над объектом – то, что выдала нейронка, если это число меньше 0.5 – это кот, если больше, то это собака. И чем больше разница, тем больше она уверена в своем выборе…

А людей она определяет примерно так:

(Всегда знал, что Максим - пёс)







## **Сложный пример:**

А теперь более сложная и интересная задачка, выполненная в рамках Агрохакатона 2020 года. Мы с коллегами потрудились над классификацией яблонь на зараженные и здоровое по снимку листочка.

Нам был предоставлен набор данных с 1400 экземплярами из которых 36% было заражено, все они находились в шумах: разнообразное освещение и фон, разный размер изображения, расфокусировки и другое. После того, как прослушали питч, мы принялись за работу – первым составили **план**:

1. Переобучить наиболее популярные и подходящие нейросеточки:
   1. Подготовить набор данных(preprocessing):
      * Провести аугментацию(случайный шум с нормальным распределением Гаусса);
      * Подогнать под размер входного слоя;
      * И стандартные операции для работы с изображениями(код)
   2. Запустить *tensorflow* под GPU с включенным увеличением памяти;
   3. Создать функцию оценки по f1 метрике;
   4. Инициализировать модельки:
      * Установить базовые слои из *tensorflow.keras.applications.{model\_name};* и отключить у них обучаемость;
      * Создать выходные слои с гиперпараметрами;
      * Собрать воедино;
   5. Обучить нейросети:
      * Скомпилировать их;
      * Сгенерировать базу данных для обучения;
      * Сгенерировать базу данных для валидации;
      * Реализовать метод быстрой остановки вместе с созданием сохранений(чекпоинтов);
      * Обучить и сохранить набор весов;
   6. Выявить лидеров по метрикам:

* Провести анализ графиков по основным показателям(потери, точность и т.д.);
* Выбрать победителя; рассмотреть его общую статистику; чувствительность(threshold - пороги), наиболее уверенные и неуверенные ответы.

1. Создать визуальный интерфейс для дальнейшей работы с готовыми моделями:
   1. Немного фронтэнда на *streamlit’е* и всё работает;

*Ну че погнали…*

### Первый этап

Создаём проект и подготавливаем среду – нам нужны вот эти импорты:

from tensorflow.keras.preprocessing.image import ImageDataGenerator  
from tensorflow.keras.applications import ResNet50V2, ResNet101V2, Xception, NASNetMobile  
from tensorflow.keras import backend as K  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Dropout, Flatten, Dense, MaxPooling2D  
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  
import random  
import numpy as np  
from scipy import ndimage  
from numpy.random import seed  
from tensorflow import random as random\_  
import pickle  
import tensorflow as tf

Для этого прописываем в терминале $ conda create --name <env> --file <this file> где <this file> заменяем на наш requirements.txt с гитхаба ЛИБО устанавливаем библиотеки вручную(первый способ может не сработаь)

Теперь поиграем с *preprocessing’*ом:

seed\_ = 1337 # сид для модуля рандома  
seed(seed\_)  
random\_.set\_seed(seed\_)  
  
def add\_noise(img): # просто функция добавление шума по Гауссу  
 rad = random.choice([1, 3, 3])  
 img = ndimage.gaussian\_filter(img, rad)  
 deviation = variability\_random\*random.random()  
 noise = np.random.normal(0, deviation, img.shape)  
 img += noise  
 img += variability \* random.gauss(0, 1)  
 np.clip(img, 0., 255.)  
 return img  
  
variability = 5 # величина изменчивости шума  
variability\_random = 5 # ^^^^^^^^  
batch\_size = 16 # количество изображений, которое видеокарта обрабатывает за раз(чем выше объём, тем больше доступно)  
n\_epochs = 25 # количество эпох  
lr = 1e-3 # learning rate - скорость обучения  
# аугментация базы данных:  
datagen = ImageDataGenerator(  
 preprocessing\_function=  
 add\_noise, # добавление нашего шума  
 horizontal\_flip=True, # отзеркаливание  
 fill\_mode="mirror", # режим заполнения данных за границами доступного набора - в данном случае режимом отзеркаливания dcba|abcd|dcba, где abcd - доступный кусочек  
 rotation\_range=5, # дипазон поворота для случайных вращений в градусах  
 rescale=1. / 255, # превращение значений пикселов (0..255) в число (от 0 до 1)  
  
 # далее всё в частях от изначальной информации:  
 shear\_range=0.1, # диапазон обрезки  
 zoom\_range=0.1, # диапазон приближения/отдаления  
 width\_shift\_range=0.1, # диапазон сдвига по x  
 height\_shift\_range=0.1, # диапазон сдвига по y  
 validation\_split=0.1, # какая часть пойдет в валидационный сет  
 )  
last\_layers = 7 # количество последних слоев в готовых моделях, которые будут заменены на собственные обучаемые слои

Запускаем *tensorflow* на видеокарту:

gpus = tf.config.experimental.list\_physical\_devices('GPU') # все доступные физические GPU  
if gpus: # если они есть, попробовать включить на каждом увеличение памяти  
 try:  
 for gpu in gpus:  
 tf.config.experimental.set\_memory\_growth(gpu, True)  
 except RuntimeError as e:  
 print(e)

Метрики для оценки работы сети:

def f1(y\_true, y\_pred): # метрика F1  
 true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))  
 possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))  
 recall = true\_positives / (possible\_positives + K.epsilon())  
 predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))  
 precision = true\_positives / (predicted\_positives + K.epsilon())  
 return 2\*((precision\*recall)/(precision+recall+K.epsilon()))  
def soft\_f1(y, y\_hat): # метрика F1   
 y = tf.cast(y, tf.float32)  
 y\_hat = tf.cast(y\_hat, tf.float32)  
 tp = tf.reduce\_sum(y\_hat \* y, axis=0)  
 fp = tf.reduce\_sum(y\_hat \* (1 - y), axis=0)  
 fn = tf.reduce\_sum((1 - y\_hat) \* y, axis=0)  
 soft\_f1 = 2 \* tp / (2 \* tp + fn + fp + 1e-16)  
 cost = 1 - soft\_f1  
 macro\_cost = tf.reduce\_mean(cost)  
 return macro\_cost

Функция создания сеточки из базовых слоёв, при этом выходные слои заменяются:

def make\_net(base):  
 base = base(input\_shape=img\_shape + (3,), include\_top=False, weights="imagenet")  
 for layer in base.layers[:-last\_layers]:  
 layer.trainable = False  
 head = base.output  
 # head = MaxPooling2D(pool\_size=(7, 7))(head)  
 head = Flatten()(head)  
 head = Dense(64, activation='relu')(head)  
 head = Dropout(0.5)(head)  
 head = Dense(1, activation='sigmoid')(head)  
 model = Model(inputs=base.input, outputs=head)  
 return model

Мы будем проходить по четырем из самых популярных сеточек из *keras.application* для классификации изображений(img\_shape подгоняем, т.к. число входных нейронов у них разное):

if \_\_name\_\_ == '\_\_main\_\_':  
 for Base, img\_shape in [(NASNetMobile, (224, 224)), (ResNet50V2, (331, 331)), (ResNet101V2, (331, 331)), (Xception, (331, 331))]:  
 model = make\_net(Base)  
 model.compile(loss="binary\_crossentropy",  
 optimizer="adam",  
 metrics=['accuracy', f1])

Генерируем train набор данных:

train\_generator = datagen.flow\_from\_directory(  
 'dataset', # this is the target directory  
 target\_size=img\_shape, # all images will be resized  
 batch\_size=batch\_size,  
 class\_mode="binary",  
 subset="training") # since we use binary\_crossentropy loss, we need binary labels

И valid набор тоже:

val\_generator = datagen.flow\_from\_directory(  
 'dataset', # this is the target directory  
 target\_size=img\_shape, # all images will be resized  
 batch\_size=batch\_size,  
 class\_mode='binary',  
 subset="validation") # since we use binary\_crossentropy loss, we need binary labels

Реализуем метод ранней остановки:

es = EarlyStopping(monitor="val\_f1", mode='max', patience=5, restore\_best\_weights=True)  
mcp = ModelCheckpoint(f'models/model-{Base.\_\_name\_\_}-' '{epoch:03d}.h5', monitor='val\_f1', mode='max')

Кормим нашу нейроночку данными…

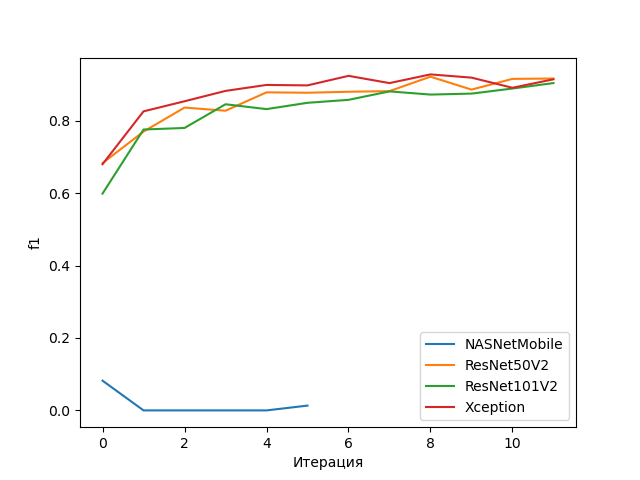
history = model.fit(  
 train\_generator,  
 steps\_per\_epoch=train\_generator.samples // batch\_size,  
 epochs=n\_epochs,  
 validation\_data=val\_generator,  
 validation\_steps=val\_generator.samples // batch\_size,  
 callbacks=[es, mcp])

Вот мы и обучили её, теперь нужно сохранить:

model.save(f'models/model-{Base.\_\_name\_\_}.h5') # always save your weights after training or during training  
pickle.dump(history.history, open(f"stories/history-{Base.\_\_name\_\_}.pkl", 'wb'))

Нужно провести анализ и выявить победителя! Для начала оценим все сразу:

import pickle  
from matplotlib import pyplot as plt  
  
all\_models = ["NASNetMobile", "ResNet50V2", "ResNet101V2", "Xception"]  
metrics = {}  
for model in all\_models:  
 logs = pickle.load(open(f"stories/history-{model}.pkl", 'rb'))  
 for metric, values in logs.items():  
 if metric not in metrics:  
 metrics[metric] = {}  
 metrics[metric][model] = values  
for metric, models in metrics.items():  
 plt.xlabel("Итерация")  
 plt.ylabel(metric)  
 for model\_name, data in models.items():  
 plt.plot(data, label=model\_name)  
 plt.legend()  
 plt.savefig(f"plots/{metric}.png")

Вот один из полученных графиков – по метрике f1. **NASNetMobile** сильно отстаёт от остальных, аномально сильно, поэтому делаем вывод, что она выпала из обучения(а т.к. на момент написания текста к работе мне лень переобучать, мы просто пропустим её. На самом хакатоне всё было нормально, но и там она была на последнем месте по точности).

Лучшие показатели у **Xception**, в конечном варианте мы его и использовали(правда в плане мощностей для feedforward’а он самый прожорливый).

Так как смысла оценивать аутсайдеров нет, рассмотрим подробнее лишь **Xception:**

В **data\_analysis.py:**

import os  
import shutil  
from sklearn import metrics  
from matplotlib import pyplot as plt  
import seaborn as sns  
import json  
import numpy as np  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
all\_models = [ # "NASNetMobile", "ResNet50V2", "ResNet101V2", # мы не берём аутсайдеров!  
 "Xception"]  
sns.set\_theme()  
sns.set\_style("white")  
datagen = ImageDataGenerator(rescale=1 / 255)

Запускаем сеть:

for model\_name, shape in zip(all\_models, [224, 331, 331, 331]):  
 model\_path = f"models/model-{model\_name}.h5"  
 test\_generator = datagen.flow\_from\_directory(  
 "dataset",  
 target\_size=(shape, shape),  
 shuffle=False,  
 class\_mode='binary',  
 )

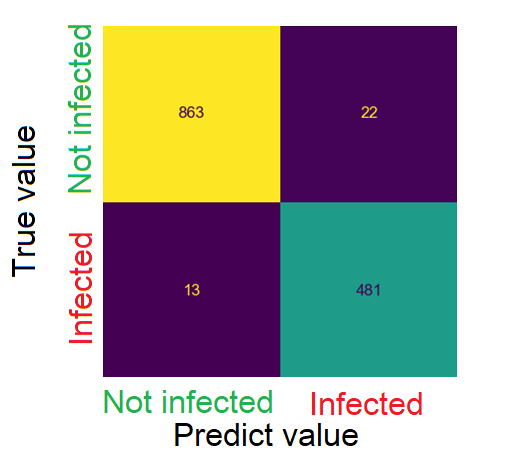
plots\_dir = f"plots/{model\_name}"  
os.makedirs(plots\_dir, exist\_ok=True)  
pred\_path = f"predictions/predictions-{model\_name}.json"

# Делаем предсказания, если таковых нет:  
if os.path.exists(pred\_path):  
 predictions = json.load(open(pred\_path))

else:  
 import tensorflow as tf  
 from tensorflow.keras.models import load\_model

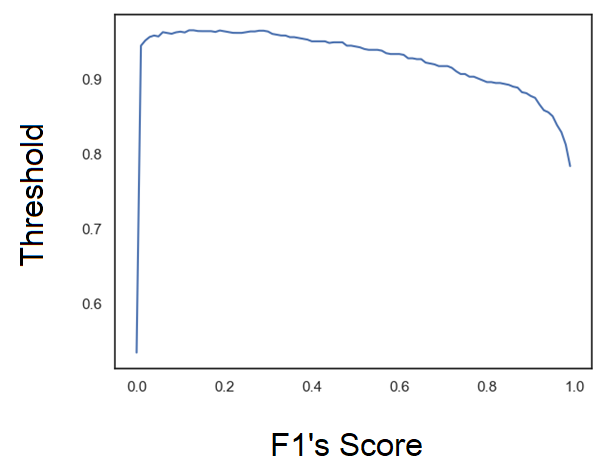
gpus = tf.config.experimental.list\_physical\_devices('GPU')  
 for gpu in gpus:  
 tf.config.experimental.set\_memory\_growth(gpu, True)  
 model = load\_model(model\_path, custom\_objects={'f1': metrics.f1\_score})  
 predictions = [float(x) for x in model.predict(test\_generator, verbose=True)]  
 json.dump(predictions, open(pred\_path, 'w'))

Делаем матрицу неточностей:

conf = metrics.confusion\_matrix([int(x > best\_threshold) for x in predictions], test\_generator.classes)  
conf = metrics.ConfusionMatrixDisplay(confusion\_matrix=conf, display\_labels=[0, 1])  
conf.plot()  
plt.axis("off")  
plt.gca().images[-1].colorbar.remove()  
plt.savefig(f"{plots\_dir}/confmatrix.png")

Записываем пороги:

ts = []  
f1s = []  
best\_f1 = 0  
best\_threshold = 0  
for i in range(100):  
 threshold = i / 100  
 ts.append(threshold)  
 y = [int(x > threshold) for x in predictions]  
 f1 = metrics.f1\_score(y, test\_generator.classes)  
 if f1 > best\_f1:  
 best\_f1 = f1  
 best\_threshold = threshold  
 f1s.append(f1)  
plt.plot(ts, f1s)  
plt.savefig(f"{plots\_dir}/threshold.png")



Выясняем, что оптимальный порог около 0.8.

Выбираем по пять наиболее уверенных и по пять наиболее неправильных:

def write\_images(out\_dir, indexes):  
 for index in indexes:  
 fn = test\_generator.filenames[index]  
 in\_file = f"dataset/{fn}"  
 os.makedirs(f"{plots\_dir}/{out\_dir}", exist\_ok=True)  
 out\_file = f"{plots\_dir}/{out\_dir}/{os.path.basename(fn)}"  
 shutil.copy2(in\_file, out\_file)

model\_predictions = np.array([int(x > best\_threshold) for x in predictions])  
indices = lambda z: [(y, i) for i, y in enumerate(z)]  
unindices = lambda z: [i for x, i in z]  
most\_confident\_positive = unindices(sorted(indices(predictions))[-5:])  
most\_confident\_negative = unindices(sorted(indices(predictions))[:5])  
wrong = model\_predictions != test\_generator.classes  
false\_positive = np.where(np.logical\_and(test\_generator.classes == 1, wrong))[0].tolist()[:5]  
false\_negative = np.where(np.logical\_and(test\_generator.classes == 0, wrong))[0].tolist()[:5]

write\_images("most\_confident/positive", most\_confident\_positive)  
write\_images("most\_confident/negative", most\_confident\_negative)  
write\_images("wrong/positive", false\_positive)  
write\_images("wrong/negative", false\_negative)

Я не фитопатолог, так что оценивать наиболее уверенные и неверные я не буду. Но они у нас есть! Второй этап именно этой работы выполнял не я, впрочем, для раскрытия темы он и не нужен, т.к. там представлена только реализация интерфейса.

Заключение

Искусственные нейронные сети получили большое распространение при решении многих задач. Вместе с тем изучение большинства их возможностей находится в экспериментальной стадии. Нейросетевые технологии не должны рассматриваться как универсальное средство решения всех интеллектуальных задач. Их применение оправдано в тех областях, в которых существует значительное число однотипных примеров, отражающих скрытые взаимосвязи.

Нейросетевые технологии предназначены для решения плохо формализованных задач. Такого рода технологии используются для распознавания каких-либо событий или предметов. С их помощью можно воспроизвести многочисленные связи между множеством объектов. Принципиальное отличие искусственных нейросетей от обычных программных систем, состоит в том, что они не требуют программирования. Они сами настраиваются, т.е. обучаются тому, что требуется пользователю.

А также НС не так сложны, как некоторые считают. Их можно довольно легко понять, прочитав пару статей. Научиться создавать свои НС с нуля незнающему программирование человеку можно меньше чем за месяц. Но я не успел выполнить практическую цель полностью, поскольку неверно распределил время на выполнение работы. И теперь, осознав свои ошибки, я буду развивать данный проект намного быстрее, чем ранее.

Реквизиты

**Бельков Руслан Марсельевич:**

**вк***-* https://vk.com/crosshairtemp;

**почта** *-* [ruslan.belckov@yandex.ru](mailto:ruslan.belckov@yandex.ru);

**github** - <https://github.com/DanteTemplar/NPK> - вся работа лежит там;

Cписок литературы

* Mohamad H.Hassoun. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, Massachusetts, 1995.
* С.Хайкин. Нейронные сети: полный курс. 2-е изд. М., "Вильямс", 2006.
* S.Haykin. Neural Networks and Learning Machines. 3rd Edition. Pearson, 2018.
* А.А.Ежов, С.А.Шумский. Нейрокомпьютинг и его применения в экономике и бизнесе. М., МИФИ, 1998. Электронная версия книги находится здесь.
* Д.А.Тархов. Нейронные сети. Модели и алгоритмы. М., Радиотехника, 2005. (Научная серия "Нейрокомпьютеры и их применение", ред. А.И.Галушкин. Кн.18.)
* А.Н.Васильев, Д.А.Тархов. Нейростевое моделирование. Принципы. Алгоритмы. Приложения. СПб.: Изд-во Политехн. Ун-та, 2009. ISBN 978-5-7422-2272-9
* Л.Г.Комарцова, А.В.Максимов. Нейрокомпьютеры. М., Изд-во МГТУ им.Баумана, 2004.
* А.И.Галушкин. Нейронные сети. Основы теории. М., Горячая линия - Телеком, 2010.
* В.А.Головко. Нейронные сети: обучение, организация и применение. М., ИПРЖР, 2001.
* Г.Э.Яхъяева. Основы теории нейронных сетей. Интернет-университет информационных технологий, изд-во "Открытые системы".
* В.В.Круглов, М.И.Дли, Р.Ю.Голунов. Нечёткая логика и искусственные нейронные сети. Физматлит, 2001.
* C.C.Aggarwal. Neural Networks and Deep Learning. A Textbook. Springer International Publishing
* AG, 2018. DOI 10.1007/978-3-319-94463-0 ISBN 978-3-319-94462-3
* К.В.Воронцов. Машинное обучение. Курс лекций.
* С.А.Шумский. Машинный интеллект. Очерки по теории машинного обучения и
* искусственного интеллекта. М., РИОР, 2019. DOI: 10.29039/02011-1
* Тема 1.1. Основные понятия. Типология задач, решаемых методами машинно