**МКОУ Лодейнопольская средняя общеобразовательная школа № 2 с углублённым изучением отдельных предметов**

# Проект

# Разработка глубокой нейронной сети для автоматизации распознавания отходов и их сортировки.

**Автор проекта:**

Ученик 10Б класса

Прошичев Александр

**Руководитель проекта:**

Драчёва Елена Рудольфовна

Лодейное Поле

2020

# 1.Вступление

- **На сколько актуален мой проект?**

  На данный момент наша планета невероятно загрязнена и с этим надо что-то делать. Мусор можно найти повсюду, в воздухе огромная концентрация вредных веществ, моря и океаны забиты отходами, что убивает живых существ – это последствия человеческого развития, а именно высасывания всего того, что удовлетворяет потребности людей и поспособствует увеличению комфорта. В наше время наука динамично развивается, поэтому для решения такого типа проблем можно использовать технологии. Как раз, поэтому я решил создать нейронную сеть для классификации отходов. Данный проект направлен именно на сортировку мусора, ведь автоматизировав такое рутинное дело, можно будет направить свои силы на что-нибудь другое, что-нибудь такое, что сделать планету чище. Поэтому, я считаю, что проект очень актуален, ведь продукт поможет делать более точно и быстро то, над чем большая группа людей занималась бы годами. Также в себе он носит использование новейших технологий, а именно нейронных сетей, которые только входят в оборот, но уже набрали огромную популярность благодаря своей эффективности.

- **Каков объект исследований?**

  Язык программирования Python и его библиотека Keras.

- **Каков предмет исследований?**

  Нейронные сети.

- **Какая проблема проекта?**

  Сортировка мусора с помощью технологий нейронных сетей.

- **Какая цель моего проекта?**

  Создать собственную глубокую нейронную сеть для автоматизации распознания отходов и их сортировки.

- **Какие задачи проекта были поставлены для достижения результата?**

  1. Изучить тему распределения и сортировки отходов.
  2. Изучить технологию нейронных сетей.
  3. Подготовить данные, на которых нейронная сеть будет обучаться и тестироваться.
  4. Обучить и протестировать полученную модель.

# 2.Теоретическая часть.

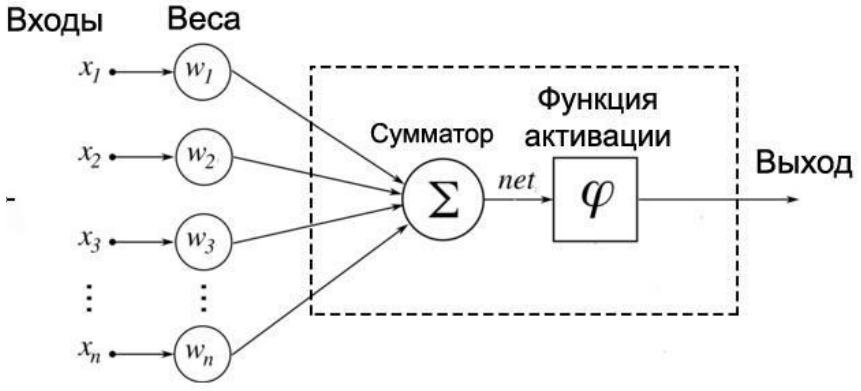
2.1) Понятие и структура нейронной сети

[**Нейронная сеть** — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямиком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти.][2] или же [**Нейронная сеть -** это машинная интерпретация мозга человека, в котором находятся миллионы нейронов, передающих информацию в виде электрических импульсов.][2].

Основные задачи, которые могут выполнять нейронные сети: **классификация** и прогноз.

[**Классификация** (от лат. classis — разряд и facere — делать) — особый случай применения логической операции деления объёма понятия, представляющий собой некоторую совокупность делений (деление некоторого на , деление этих видов и так далее).][19]

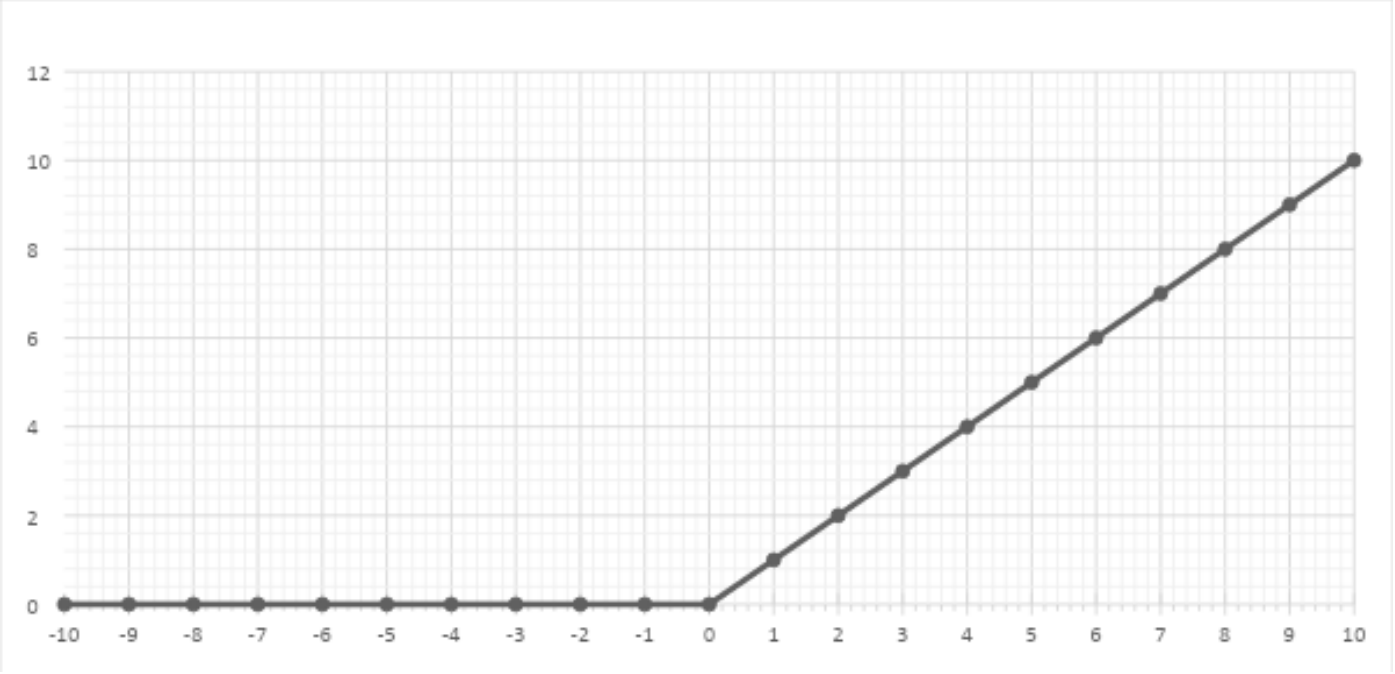
В 1956 году Маккалок Дж. и Питтс У. предложили модель искусственного нейрона, который в будущем станет основополагающим в любой нейросети.




[Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше.][2]

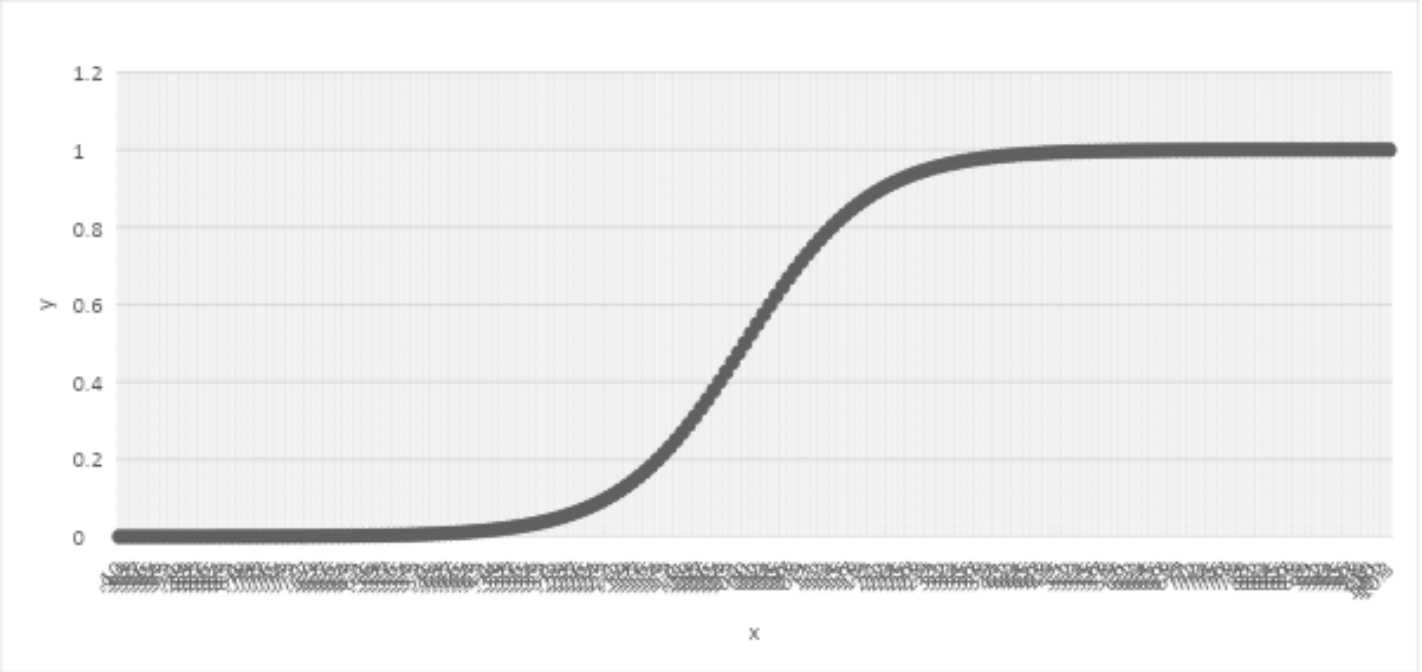
Ко входу нейрона подключаются так называемые **синапсы**. [**Синапс** - это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому.][2] По итогу синапсы подходят к **сумматору**. **Сумматор** складывает произведения весов синапсов со значениями, полученными от нейронов, от которых пришёл сигнал. К примеру, в приведённой выше схеме нейрона выход из сумматора будет равен $x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \ldots + x_n * w_n$. Этот выход проходит через **функцию активации**. [**Функция активации** — это способ нормализации входных данных. То есть, если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне. ][2] После чего значение функции является выходом нейрона и передаётся дальше. Рассмотрим некоторые из них:

Функция активации **RELU** (от англ. **rectified linear unit**) – функция-выпрямитель, формула которой $y = \max(0, x)$. Служит для аппроксимации значений. Если сумматор нейрона с такой функцией активации выдаст отрицательное значение, то нейрон не возбудится и никак не повлияет на следующие нейроны, то есть его выход будет равен 0.




Функция активации **SIGMOID** – логистическая функция с диапазоном (0; 1), формула которой $y = 1/(1 + e^{-x})$. Служит для нормализации данных до

заданного диапазона и часто играет роль вероятности.




Нейроны делятся на три основные группы. Входные, скрытые и выходные. [Входной слой, который получает информацию, n скрытых слоев, которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные и выходные данные. В случае входного нейрона – это одни и те же данные. В остальных, на вход попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации][2]

2.2) Обучение нейронной сети

Но такая сеть будет выдавать правильные ответы только при правильных весах на синапсах. В связи с чем, нужно задать эти самые веса. Этим процессом и называет **обучение нейронной сети**.
[Процесс **обучения нейронной сети** заключается в подстройке ее внутренних параметров под конкретную задачу. Алгоритм работы нейронной сети является **итеративным**, его шаги называют **эпохами** или циклами. **Эпоха** - несколько итерации в процессе **обучения**, включающих предъявление всех примеров из определённого **сета** и, возможно, проверку качества **обучения** на контрольном **сете**.][8] [**Сет** — это последовательность данных, которыми оперирует нейронная сеть.][2] [**Итерация** – это своеобразный счетчик, который увеличивается каждый раз, когда нейронная сеть проходит один сет.][2] [При инициализации нейронной сети эпоха устанавливается в 0 и имеет потолок, задаваемый вручную. Чем больше эпоха, тем лучше натренирована сеть и

соответственно, ее результат. Эпоха увеличивается каждый раз, когда мы проходим все возможные итерации][2]. Однако, такая сеть не будет обучатся. Она будет только бегать по сетам и выдавать неправильные ответы. Ведь мы не изменяем её веса. Во-первых, нужно найти **ошибку** и «указать» программе на неё. [**Ошибка** — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так.][2] Вычисления этой величины могут быть разными. [Простейший и самый распространённый пример оценки — сумма квадратов расстояний от выходных сигналов сети до их требуемых значений (см. рисунок ниже)][9]

$$E_p = \frac{1}{2}\sum_j (t_{pj} - y_{pj})^2 , \qquad (2.8)$$

где $E_p$ — величина функции ошибки для образа $p$;

$t_{pj}$ — желаемый выход нейрона $j$ для образа $p$;

$y_{pj}$ — активированный выход нейрона $j$ для образа $p$.

Существует несколько способов обучение. Но в данном проекте будет рассмотрен только один – **Обучение с учителем**.
[**Обучение с учителем** — это тип тренировок присущий таким проблемам как регрессия и классификация (им мы и воспользовались в примере, приведенном выше). Иными словами, здесь вы выступаете в роли учителя, а НС в роли ученика. Вы предоставляете входные данные и желаемый результат, то есть ученик, посмотрев на входные данные поймет, что нужно стремиться к тому результату, который вы ему предоставили.][13]

[В качестве примера, обучим сеть на объекте (1,1,0), таким образом, значения x1 и x2 равны 1, а t равно 0. Построим график зависимости ошибки E от действительного ответа y, его результатом будет парабола. Минимум параболы соответствует ответу y, минимизирующему E. Если тренировочный объект один, минимум касается горизонтальной оси, следовательно, ошибка будет нулевая и сеть может выдать ответ y равный ожидаемому ответу t, следовательно, задача преобразования входных значений в выходные может быть сведена к задаче оптимизации, заключающейся в поиске функции, которая даст минимальную ошибку.

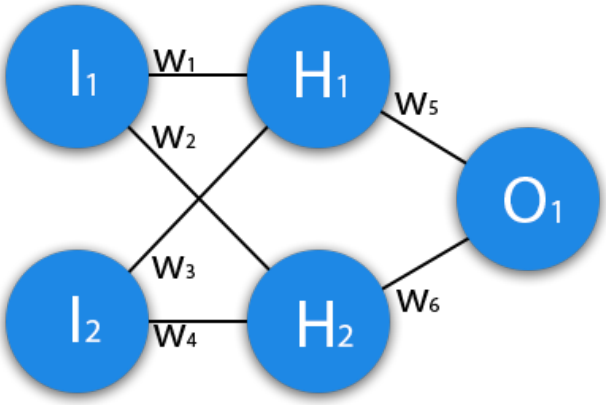
График ошибки для нейрона с линейной функцией активации и одним тренировочным объектом

В таком случае, выходное значение нейрона — взвешенная сумма всех его входных значений: y=x1w1+x2w2, где w1 и w2 — веса на ребрах, соединяющих входные вершины с выходной. Следовательно, ошибка зависит от весов ребер, входящих в нейрон. И именно это нужно менять в процессе обучения. Распространенный алгоритм для поиска набора весов, минимизирующего ошибку — градиентный спуск. Метод **обратного распространения ошибки** используется для вычисления самого "крутого" направления для спуска.][10]

Когда мы нашли ошибку, с её помощью нужно изменить веса всей нейронной сети и на этот процесс есть много методов. Мы рассмотрим один из них.

[**Adam** (adaptive moment estimation) - оптимизационный алгоритм, сочетающий в себе и идею накопления движения и идею более слабого обновления весов для типичных признаков. Накапливает значения градиента. Кроме того, мы хотим знать, как часто градиент изменяется. Авторы алгоритма предложили для этого оценивать ещё и среднюю нецентрированную дисперсию. Важное отличие состоит в начальной калибровке, но есть проблема: если задать нулевое начальное значение, то они будут долго накапливаться, особенно при большом окне накопления, а какие-то изначальные значения — это ещё два гиперпараметра. Никто не хочет ещё два гиперпараметра, так что мы искусственно увеличиваем и на первых шагах][12]
Перейдём к метода **обратного распространения ошибки**, зная его все составляющие.

[**Обратное распространение ошибки** (МОР) — это способ обучения нейронной сети. Цели обратного распространения просты: отрегулировать каждый вес пропорционально тому, насколько он способствует общей ошибке. Если мы будем итеративно уменьшать ошибку каждого веса, в конце концов у нас будет ряд весов, которые дают хорошие прогнозы.][11]



[Так как мы уже подсчитали результат НС и ее ошибку, то мы можем сразу приступить к МОРу. Как я уже упоминал ранее, алгоритм всегда начинается с выходного нейрона. В таком случае давайте посчитаем для него значение δ (дельта) по формуле 1.

$$1)\ \delta_O = (OUT_{ideal} - OUT_{actual}) * f'(IN)$$

$$2)\ \delta_H = f'(IN) * \sum(w_i * \delta_i)$$

Так как у выходного нейрона нет исходящих синапсов, то мы будем пользоваться первой формулой (δ output), следственно для скрытых нейронов мы уже будем брать вторую формулу (δ hidden). Тут все достаточно просто: считаем разницу между желаемым и полученным результатом и умножаем на производную функции активации от входного значения данного нейрона. Прежде чем приступить к вычислениям я хочу обратить ваше внимание на производную. Во-первых, как это уже, наверное, стало понятно, с МОР нужно использовать только те функции активации, которые могут быть дифференцированы. Во вторых чтобы не делать лишних вычислений, формулу производной можно заменить на более дружелюбную и простую формула вида:

$$f'(IN) = \begin{cases} f_{sigmoid} = (1 - OUT) * OUT \\ f_{tangh} = 1 - OUT^2 \end{cases}$$

Таким образом наши вычисления для точки O1 будут выглядеть следующим образом.
На этом вычисления для нейрона O1 закончены. Запомните, что после подсчета дельты нейрона мы обязаны сразу обновить веса всех исходящих синапсов этого нейрона. Так как в случае с O1 их нет, мы переходим к нейронам скрытого уровня и делаем тоже самое за исключение того, что формула подсчета дельты у нас теперь вторая и ее суть заключается в том, чтобы умножить производную функции активации от входного значения на

сумму произведений всех исходящих весов и дельты нейрона с которой этот синапс связан. Но почему формулы разные? Дело в том, что вся суть МОР заключается в том, чтобы распространить ошибку выходных нейронов на все веса НС. Ошибку можно вычислить только на выходном уровне, как мы это уже сделали, также мы вычислили дельту, в которой уже есть эта ошибка. Следственно теперь мы будем вместо ошибки использовать дельту, которая будет передаваться от нейрона к нейрону. Теперь нам нужно найти градиент для каждого исходящего синапса. Здесь обычно вставляют 3 этажную дробь с кучей производных и прочим математическим адом, но в этом и вся прелесть использования метода подсчета дельт, потому что в конечном счете ваша формула нахождения градиента будет выглядеть вот так:

$$GRAD_B^A = \delta_B * OUT_A$$

Здесь точка A это точка в начале синапса, а точка B на конце синапса. Таким образом мы можем подсчитать градиент w5. Сейчас у нас есть все необходимые данные чтобы обновить вес w5 и мы сделаем это благодаря функции МОР которая рассчитывает величину на которую нужно изменить тот или иной вес и выглядит она следующим образом:

$$\Delta w_i = E * GRADw + \alpha * \Delta w_{i-1}$$

Настоятельно рекомендую вам не игнорировать вторую часть выражения и использовать момент так как это вам позволит избежать проблем с локальным минимумом.

Здесь мы видим 2 константы о которых мы уже говорили, когда рассматривали алгоритм градиентного спуска: E (эпсилон) — скорость обучения, α (альфа) — момент. Переводя формулу в слова получим: изменение веса синапса равно коэффициенту скорости обучения, умноженному на градиент этого веса, прибавить момент, умноженный на предыдущее изменение этого веса (на 1-ой итерации равно 0). В таком случае давайте посчитаем изменение веса w5 и обновим его значение прибавив к нему Δw5.][13]

[**Переобучение**, или чрезмерно близкая подгонка - излишне точное соответствие нейронной сети конкретному набору обучающих примеров, при котором сеть теряет способность к обобщению.
Переобучение возникает в случае слишком долгого обучения, недостаточного числа обучающих примеров или переусложненной структуры нейронной сети. Переобучение связано с тем, что выбор обучающего (тренировочного) множества является случайным. С первых шагов обучения происходит уменьшение ошибки. На последующих шагах с целью уменьшения ошибки (целевой функции) параметры подстраиваются под особенности обучающего множества. Однако при этом происходит "подстройка" не под общие закономерности ряда, а под особенности его части - обучающего подмножества. При этом точность прогноза уменьшается. Один из вариантов борьбы с переобучением сети - деление обучающей выборки на два множества (обучающее и тестовое). На обучающем множестве происходит обучение нейронной сети. На тестовом множестве осуществляется проверка построенной модели. Эти множества не должны пересекаться. С каждым шагом параметры модели изменяются, однако постоянное уменьшение значения целевой функции происходит именно на обучающем множестве. При разбиении множества на два мы можем наблюдать изменение ошибки прогноза на тестовом множестве параллельно с наблюдениями над обучающим множеством. Какое-то количество шагов ошибки прогноза уменьшается на обоих множествах. Однако на определенном шаге ошибка на тестовом множестве начинает возрастать, при этом ошибка на обучающем множестве продолжает уменьшаться. Этот момент считается концом реального или настоящего обучения, с него и начинается переобучение.][8]

## 2.3) Понятие свёрточных нейронных сетей

Теперь мы умеем обучать нейронные сети. Но есть большая проблема: мы хотим обрабатывать цветные изображения (например, с разрешением w * h), но для этого придётся создать w * h* 3 входных нейронов, что очень много. Для этой задачи примитивные сети не подойдут, надо рассмотреть более сложные по своей структуре (**Глубокие нейронные сети**).
[**Глубокая нейронная сеть** — это **нейронная сеть** с несколькими скрытыми слоями.][16]
Типы глубоких сетей: рекуррентные и свёрточные.
[**Глубокое обучение** (deep learning) — это подмножество методов машинного обучения, области изучения и создания машин, которые могут обучаться (иногда с целью достичь уровня искусственного интеллекта).

Глубокое обучение используется в промышленности для решения практических задач в самых разных областях, таких как компьютерное зрение (изображения), обработка естественного языка (текст) и автоматическое распознавание речи. Проще говоря, глубокое обучение — это подмножество методов машинного обучения, главным образом основанных на применении искусственных нейронных сетей, которые представляют класс алгоритмов, подражающих человеческому мозгу][15]

Для обработки изображения на понадобится именно **свёрточная нейронная сеть**.

[**Свёрточная нейронная сеть** (англ. *convolutional neural network*, *CNN*) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения (англ. *deep learning*). Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании **свёрточных слоёв** (англ. *convolution layers*) и субдискретизирующих слоёв (англ. *subsampling layers* или англ. *pooling layers*, **слоёв подвыборки**). Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

Название архитектура сети получила из-за наличия операции **свёртки**, суть которой в том, что каждый фрагмент изображения умножается на матрицу **ядро свёртки** поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.][17]

[В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют **ядром свёртки**. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой

матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков (англ. feature map). Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая элементы изображения (например, линии и дуги под разными углами). При этом такие ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной (много независимых карт признаков на одном слое). Также следует отметить, что при переборе слоя матрицей весов её передвигают обычно не на полный шаг (размер этой матрицы), а на небольшое расстояние. Так, например, при размерности матрицы весов 5×5 её сдвигают на один или два нейрона (пикселя) вместо пяти, чтобы не «перешагнуть» искомый признак.][16]

[**Свёртка функций** — операция в функциональном анализе, которая при применении к двум функциям $f$ и $g$, возвращает третью функцию, соответствующую взаимно корреляционной функции $f(x)$ и $g(-x)$. Операцию свертки можно интерпретировать как «схожесть» одной функции с отражённой и сдвинутой копией другой. Понятие свёртки обобщается для функций, определённых на произвольных измеримых пространствах и может рассматриваться как особый вид интегрального преобразования.][18]

[**Слой свёртки** (англ. *convolutional layer*) — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, **ядро свёртки** которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента). Весовые коэффициенты **ядра свёртки** (небольшой матрицы) неизвестны и устанавливаются в процессе обучения.

Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении. Так, например, если исходное изображение имеет размерность 100×100 пикселей по трём каналам (это значит 30000 входных нейронов), а свёрточный слой использует фильтры с ядром 3x3 пикселя с выходом на 6 каналов, тогда в процессе обучения определяется только 9 весов ядра, однако по всем сочетаниям каналов, то есть 9×3×6 =162, в таком случае данный слой требует нахождения только 162 параметров, что существенно меньше количества искомых параметров полносвязной нейронной сети.][16]

[**Слой подвыборки** (иначе, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей

(обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция подвыборки позволяет существенно уменьшить пространственный объём изображения. Подвыборка интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой подвыборки, как правило, вставляется после слоя свёртки перед слоем следующей свёртки.

Кроме подвыборки с функцией максимума можно использовать и другие функции — например, среднего значения или *L2-нормирования*. Однако практика показала преимущества именно подвыборки с функцией максимума, который включается в типовые системы.

В целях более агрессивного уменьшения размера получаемых представлений, всё чаще находят распространение идеи использования меньших фильтров или полный отказ от слоёв подвыборки.][16]

2.4) Сортировка мусора

Теперь нужно выбрать правильную структуру нейронной сети. Задать её входные и выходные данные. Для этого определим, на какие классы будут делится отходы.

Их можно разделять на огромное количество классов, но есть самые базовые.

[Большинство отходов – вторсырье, которое может быть повторно переработано. Переработать можно пластик, бумагу, металл, стекло.][1], кроме того [Из всех отходов жизнедеятельности человека более 30 % составляет органика.][4] Что является основанием для включения органического мусора в список распознаваемых отходов разрабатываемой нейронной сети. Таким образом, будут использоваться 5 классов, представленных на изображении ниже.

В связи с выше сказанным, входными данными нейронной сети будут являться фотографии (изображения) отходов, а выходными – наименование класса отходов, которые нейросеть должна распознать.

# 3. Практическая часть.

Для разработки индивидуального проекта был использован язык программирования Python[22], его библиотека Keras[21], онлайн среда разработки Google Colaboratory[20].

## 3.1) Подготовка учебного материала для нейронной сети

Готовим данные для обучения и тестирования будущей нейронной сети.

Для максимизации точности результаты выдаваемых ответов произведём деление фотографий на 3 класса: обучающие (learn), проверяющие (validations) и тестовые (test). Программа будет обучится на первых, проверятся между эпохами на вторых, и под конец обучения будет произведено контрольное тестирование сети. На обучение данной нейронной сети было решено выделить 25.000 фотографий: 15.000 – learn, 5.000 – val, 5.000 – test.


Так как всего классов 5, то на каждый класс 3.000 – learn, 1.000 – val, 1.000 – test.


Все изображения были взята с Kaggle[23], Яндекс.Картинки[24] или ImageNet[25].

Для содержания изображений и их использования используется облачное хранилище Google Drive[26]

## 3.2) Реализация нейронной сети

1, 2, 3 блок кода: подключение всех нужных библиотек, функций и Облачного хранилища Google Drive[26] для реализации и использования нейронной сети.

```
[ ]  from tensorflow.python.keras.models import Sequential, load_model
     from tensorflow.python.keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
     from google.colab import files, drive
     from tensorflow.python.keras.preprocessing import image
     from IPython.display import Image
     import matplotlib.pyplot as plt
     import PIL
     import numpy as np
```

```
[ ]  from PIL import ImageFile
     ImageFile.LOAD_TRUNCATED_IMAGES = True
```

```
[ ]  drive.mount('/content/here', force_remount=True)
```

4 блок кода: в переменные train, val и test сохраняем пути нахождения папок в Google Drive[26] с тренировочными, проверяющими и тестирующими данными соответственно. Далее указан размер изображений, под который все будут сжиматься, вид входных данных (длина изображения, ширина изображения, три канала цвета), количество эпох, количество подаваемых изображений за одну итерацию.

```
[ ]  train = 'here/My Drive/learn'
     val = 'here/My Drive/val'
     test = 'here/My Drive/test'
     w, h = 150, 150
     input_S = (w, h, 3)
     epochs = 35
     batch_size = 15
```

5 блок кода: построение архитектуры модели.

Создаём модель и добавляем в неё слои:

Conv2D – слой свёртки, где (3, 3) – размер ядра свёртки;

Activation – функция активации;

MaxPooling2D – слой подвыборки;

Flatten – перевод картинки в линейный массив чисел;

Dense – добавление скрытого слоя из 64 нейронов;

Dropout – вероятность отключения каждого нейрона вовремя обучения против переобучения.

```
[ ]  model = Sequential()
     model.add(Conv2D(32, (3, 3), input_shape=input_S))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(Conv2D(32, (3, 3)))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(Conv2D(64, (3, 3)))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(Flatten())
     model.add(Dense(64))
     model.add(Activation('relu'))
     model.add(Dropout(0.5))
     model.add(Dense(5))
     model.add(Activation('sigmoid'))
```

Каждый из 5 нейронов в последнем слою отвечает за отдельный класс из представленных. Функция активации Sigmoid выдаёт вероятность принадлежности объекта к каждому из классов.

6 блок кода: компиляция модели с использованием метода обратного распространения ошибки, оптимизатора adam и метрики аккуратности (будем визуализировать проценты успеваемости нейронной сети).

```
[ ]  model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

7 и 8 блок кода: создание ядер генераторов, которые будут автоматически подавать изображения нейронной сети. Первое ядро является обучающим: генератор с таким ядром будет брать изображение, искажать и изменять его. Это сделано из-за относительного малого количества данных. Тем самым из одного фото можно сделать бесконечно много. Второе ядро

просто передаёт изображения в почти в исходном виде. Только ради удобства обучения для нейронной сети, каждый пиксель каждого изображения переводится в диапазон [0; 1]. Это очень удобный формат для программы.

```
[ ]  learn_test = image.ImageDataGenerator(rescale=1./255,
                                           rotation_range=90,
                                           width_shift_range=0.2,
                                           height_shift_range = 0.2,
                                           zoom_range=0.2,
                                           horizontal_flip = True,
                                           vertical_flip = True,
                                           fill_mode='nearest') #...
```

```
[ ]  datagen = image.ImageDataGenerator(rescale=1./255)
```

9, 10 и 11 блок кода: создание генераторов на основе ранее созданных ядер и подключение их к облачному хранилищу, далее описание обработки изображений.

После активации данных блоков нам выдаётся информация о количестве

```
[ ]  trainsys = learn_test.flow_from_directory(
        train,
        target_size=(w, h),
        batch_size=batch_size,
        class_mode = 'categorical'
    )
```

```
Found 5000 images belonging to 5 classes.
```

```
[ ]  valsys = datagen.flow_from_directory(
        test,
        target_size = (w, h),
        batch_size=batch_size,
        class_mode='categorical'
    )
```

```
Found 5002 images belonging to 5 classes.
```

обнаруженных данных и классов.

```
Found 14998 images belonging to 5 classes.
```

Видно, что при распределении изображения были разбиты неровно, пара из них оказалась в другом наборе. Но этим можно пренебречь.

12 блок кода: обучение нейронной сети. 35 эпох обучения с 857 итерациями, между которыми происходит проверка точности в 142 итерации.

```
[ ]  model.fit_generator(
         trainsys,
         steps_per_epoch= 857,
         epochs=35,
         validation_data = valsys,
         validation_steps= 142
     )
```

В меню снизу будет указана вся нужная информация об обучении.

```
Epoch 1/35
 27/857 [..............................] - ETA: 1:15:57 - loss: 1.6113 - accuracy: 0.2099/usr/local/lib/python3.6/dist-packages/PIL/Im
  "Palette images with Transparency expressed in bytes should be "
857/857 [==============================] - 4681s 5s/step - loss: 1.4855 - accuracy: 0.3283 - val_loss: 1.4293 - val_accuracy: 0.3864
Epoch 2/35
857/857 [==============================] - 1643s 2s/step - loss: 1.4265 - accuracy: 0.3602 - val_loss: 1.3582 - val_accuracy: 0.3709
Epoch 3/35
857/857 [==============================] - 693s 809ms/step - loss: 1.3912 - accuracy: 0.3713 - val_loss: 1.4299 - val_accuracy: 0.4020
Epoch 4/35
857/857 [==============================] - 499s 583ms/step - loss: 1.3754 - accuracy: 0.3949 - val_loss: 1.2841 - val_accuracy: 0.4798
Epoch 5/35
857/857 [==============================] - 494s 576ms/step - loss: 1.3367 - accuracy: 0.4262 - val_loss: 1.2439 - val_accuracy: 0.5033
Epoch 6/35
857/857 [==============================] - 498s 581ms/step - loss: 1.3225 - accuracy: 0.4317 - val_loss: 1.2049 - val_accuracy: 0.5009
Epoch 7/35
857/857 [==============================] - 497s 580ms/step - loss: 1.2718 - accuracy: 0.4727 - val_loss: 1.1979 - val_accuracy: 0.5122
Epoch 8/35
857/857 [==============================] - 501s 584ms/step - loss: 1.2415 - accuracy: 0.4860 - val_loss: 1.1668 - val_accuracy: 0.5236
Epoch 9/35
857/857 [==============================] - 499s 582ms/step - loss: 1.2342 - accuracy: 0.4970 - val_loss: 1.1167 - val_accuracy: 0.5723
Epoch 10/35
857/857 [==============================] - 500s 584ms/step - loss: 1.1968 - accuracy: 0.5186 - val_loss: 1.1232 - val_accuracy: 0.5594
Epoch 11/35
857/857 [==============================] - 501s 585ms/step - loss: 1.1903 - accuracy: 0.5230 - val_loss: 1.1638 - val_accuracy: 0.5343
```

```
Epoch 12/35
857/857 [==============================] - 495s 577ms/step - loss: 1.1708 - accuracy: 0.5316 - val_loss: 1.2883 - val_accuracy: 0.5184
Epoch 13/35
857/857 [==============================] - 497s 580ms/step - loss: 1.1616 - accuracy: 0.5328 - val_loss: 1.1163 - val_accuracy: 0.5728
Epoch 14/35
857/857 [==============================] - 496s 579ms/step - loss: 1.1529 - accuracy: 0.5435 - val_loss: 1.0661 - val_accuracy: 0.5934
Epoch 15/35
857/857 [==============================] - 495s 577ms/step - loss: 1.1381 - accuracy: 0.5545 - val_loss: 1.1198 - val_accuracy: 0.5768
Epoch 16/35
857/857 [==============================] - 496s 579ms/step - loss: 1.1228 - accuracy: 0.5603 - val_loss: 1.1143 - val_accuracy: 0.5751
Epoch 17/35
857/857 [==============================] - 496s 579ms/step - loss: 1.1117 - accuracy: 0.5672 - val_loss: 1.0588 - val_accuracy: 0.5933
Epoch 18/35
857/857 [==============================] - 499s 583ms/step - loss: 1.1013 - accuracy: 0.5697 - val_loss: 1.1105 - val_accuracy: 0.5798
Epoch 19/35
857/857 [==============================] - 501s 585ms/step - loss: 1.1036 - accuracy: 0.5769 - val_loss: 1.0933 - val_accuracy: 0.6018
Epoch 20/35
857/857 [==============================] - 493s 575ms/step - loss: 1.0941 - accuracy: 0.5731 - val_loss: 1.0117 - val_accuracy: 0.6183
Epoch 21/35
857/857 [==============================] - 499s 582ms/step - loss: 1.0822 - accuracy: 0.5827 - val_loss: 1.1025 - val_accuracy: 0.5939
Epoch 22/35
857/857 [==============================] - 496s 579ms/step - loss: 1.0751 - accuracy: 0.5889 - val_loss: 1.0330 - val_accuracy: 0.6037
Epoch 23/35
857/857 [==============================] - 495s 578ms/step - loss: 1.0662 - accuracy: 0.5926 - val_loss: 1.1053 - val_accuracy: 0.6005
Epoch 24/35
857/857 [==============================] - 498s 581ms/step - loss: 1.0564 - accuracy: 0.5981 - val_loss: 1.0317 - val_accuracy: 0.5999
Epoch 25/35
857/857 [==============================] - 502s 586ms/step - loss: 1.0430 - accuracy: 0.6017 - val_loss: 1.0233 - val_accuracy: 0.6155
```

```
Epoch 26/35
857/857 [==============================] - 499s 583ms/step - loss: 1.0405 - accuracy: 0.6027 - val_loss: 0.9592 - val_accuracy: 0.6414
Epoch 27/35
857/857 [==============================] - 496s 579ms/step - loss: 1.0278 - accuracy: 0.6093 - val_loss: 0.9696 - val_accuracy: 0.6324
Epoch 28/35
857/857 [==============================] - 492s 574ms/step - loss: 1.0442 - accuracy: 0.6051 - val_loss: 1.0475 - val_accuracy: 0.5958
Epoch 29/35
857/857 [==============================] - 496s 579ms/step - loss: 1.0280 - accuracy: 0.6121 - val_loss: 0.9686 - val_accuracy: 0.6357
Epoch 30/35
857/857 [==============================] - 494s 576ms/step - loss: 1.0252 - accuracy: 0.6111 - val_loss: 1.0275 - val_accuracy: 0.6164
Epoch 31/35
857/857 [==============================] - 496s 579ms/step - loss: 1.0194 - accuracy: 0.6107 - val_loss: 1.0382 - val_accuracy: 0.6140
Epoch 32/35
857/857 [==============================] - 496s 579ms/step - loss: 1.0024 - accuracy: 0.6223 - val_loss: 1.0263 - val_accuracy: 0.6122
Epoch 33/35
857/857 [==============================] - 503s 587ms/step - loss: 0.9940 - accuracy: 0.6193 - val_loss: 1.0729 - val_accuracy: 0.6027
Epoch 34/35
857/857 [==============================] - 493s 576ms/step - loss: 0.9907 - accuracy: 0.6263 - val_loss: 0.9498 - val_accuracy: 0.6502
Epoch 35/35
857/857 [==============================] - 498s 581ms/step - loss: 0.9912 - accuracy: 0.6290 - val_loss: 0.9904 - val_accuracy: 0.6254
```

Как можно заметить, в 4 и 6 колонке данных указаны точности на обучающих и проверяющих наборах, и с каждой эпохой эти числа растут.

13 и 14 блок кода: запускаем итоговую аттестацию нейронной сети на тестовом наборе и видим результат: 65.78%

```
[ ]  x = model.evaluate_generator(testsys, 334)
```

```
[ ]  print(x[1] * 100)
```

```
65.7800018787384
```

Нейронная сеть обучена и готова к личной проверке. Проверим на данной фотографии:

15 блок кода: указываем массив названий в том порядке, в котором у нас шли классы при подготовке данных.

```
[ ]  result = ["Бумажные изделия", "Металл", "Органика", "Пластик", "Стекло"]
```

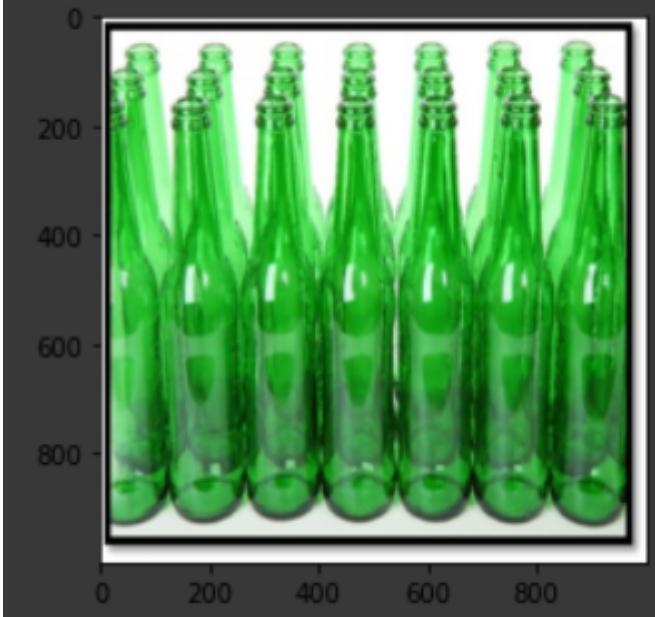
16, 17 и 18 блок кода: Мы загружаем изображение в память.

```
[ ]  x = files.upload()
```

Выводим её для того, чтобы убедится в правильности данных.

```
[16] image_file_name = 'бутылки.jpg'
     img = image.load_img(image_file_name, target_size=(1000, 1000))
     plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f3cdf157d68>
```




Загрузка прошла удачно.

Далее обрабатываем изображение. Переводя данные в пригодный для нейронной сети вид.

```
[17] res = image.load_img('бутылки.jpg', target_size=(150, 150))
     res = image.img_to_array(res)
     res = np.expand_dims(res, axis=0)
     res /= 255
```

19 блок кода: Подаём изображение на вход программе и выводим результат.

```
res = model.predict(res)
print(res)
print(result[np.argmax(res)])
```

```
[[0.12377056 0.19553475 0.06821914 0.2807692  0.3115167 ]]
Стекло
```

Выводится массив, заполненный вероятностями отношения изображения к определённым классам. Наибольшая из них имеет одинаковый индекс с классом стекла, это и ответом нашей программы.

# 4.Заключение.

В ходе индивидуальной работы я изучил тему распределения и сортировки мусора, изучил технологию нейронных сетей, подготовил данные для обучения будущей модели и, в итоге, создал, обучил и протестировал произведённую программу. Безусловно полученное ИИ не является идеальным алгоритмом.

Во-первых, алгоритм выдаёт на контрольных данных ~66% точности распределения, что очень мало для массового производства (из 50 различного мусора правильно будет сортироваться только 33).

Во-вторых, модель обучалась на чистых, качественных изображениях. При учёте того количества отходов, которое каждый день проходит через мусорные баки, можно точно утверждать, что многие изображения будут размытыми, тёмными и т.д.

В результате, мы получаем очень перспективное, но сырое изделие, которое нуждается в многочисленных переработках и оптимизациях. Данный проект направлен именно на сортировку мусора, ведь автоматизировав такое рутинное дело, можно будет направить свои силы на что-нибудь другое, что-нибудь такое, что сделать планету чище. Это изменит мир!

# 5.Вспомогательные ресурсы.

4.1) Используемая литература:


1) https://zen.yandex.ru/media/id/5b97d891b4dba900ac7c5ad7/razdelnyi-sbor-musora-kak-pravilno-sortirovat-othody-5c882a30f836be00b4fbd496

2) https://habr.com/ru/post/312450/

3) https://www.youtube.com/playlist?list=PLtPJ9lKvJ4oiz9aaL_xcZd-x0qd8G0VN_

4) https://bezotxodov-ru.turbopages.org/s/bezotxodov.ru/jekologija/utilizacija-organicheskih-othodov

5) https://www.youtube.com/watch?v=IzweQmZPFpw

6) https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

7) https://ru.wikipedia.org/wiki/Функция_активации

8) https://www.intuit.ru/studies/courses/6/6/lecture/178?page=4

9) https://ru.wikipedia.org/wiki/Метод_обратного_распространения_ошибки

10)https://neerc.ifmo.ru/wiki/index.php?title=Обратное_распространение_ошибки

11) https://neurohive.io/ru/osnovy-data-science/obratnoe-rasprostranenie/

12) https://habr.com/ru/post/318970/

13) https://habr.com/ru/post/313216/

14)https://vk.com/doc44301783_518781262?hash=b419e01b4ef7a85a7e&dl=9aaef78449bd721469

15) https://habr.com/ru/post/348028/

16) https://habr.com/ru/post/348000/

17) https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть

18) https://ru.wikipedia.org/wiki/Свёртка_(математический_анализ)

19) https://kartaslov.ru/значение-слова/классификация


4.2) Организационные сервисы:


20) https://colab.research.google.com/

21) https://keras.io/

22) https://www.python.org/

23) https://www.kaggle.com/

24) https://yandex.ru/images/

25) http://www.image-net.org/

26) https://drive.google.com/