[Компьютерное зрение 3](#_Toc136509421)

[Структура курса 3](#_Toc136509422)

[Ваши цели 3](#_Toc136509423)

[Проектная работа 4](#_Toc136509424)

[Ознакомьтесь с проектом, который нужно выполнить в конце курса. 4](#_Toc136509425)

[Описание проекта 4](#_Toc136509426)

[Инструкция по выполнению проекта 4](#_Toc136509427)

[Описание данных 4](#_Toc136509428)

[Как будут проверять проект? 4](#_Toc136509429)

[Введение 5](#_Toc136509430)

[Что вы изучите? 5](#_Toc136509431)

[Что такое изображение 5](#_Toc136509432)

[Типы изображений 5](#_Toc136509433)

[Что значит «растровое»? 5](#_Toc136509434)

[Что такое RGB? 5](#_Toc136509435)

[Что такое пиксели? 5](#_Toc136509436)

[Параметры изображения 6](#_Toc136509437)

[Как посчитать размер изображения 6](#_Toc136509438)

[Выводы 6](#_Toc136509439)

[Аугментации изображений 7](#_Toc136509440)

[OpenCV 7](#_Toc136509441)

[image = cv2.imread('Lenna.png') 7](#_Toc136509442)

[Отражение по горизонтали и вертикали 8](#_Toc136509443)

[Яркость, контраст и гамма 9](#_Toc136509444)

[Выводы 11](#_Toc136509445)

[Задача классификации изображений 12](#_Toc136509446)

[Ограничения классов 12](#_Toc136509447)

[Препроцессинг данных 12](#_Toc136509448)

[Модели компьютерного зрения 12](#_Toc136509449)

[Функции потерь и метрики качества 12](#_Toc136509450)

[Обучение модели классификации изображений 13](#_Toc136509451)

[Выводы 13](#_Toc136509452)

[Задачи детекции и сегментации 14](#_Toc136509453)

[Задача детекции объектов 14](#_Toc136509454)

[Метрики качества детекции 14](#_Toc136509455)

[Алгоритм детекции объектов 15](#_Toc136509456)

[Функция потерь в задаче детекции 16](#_Toc136509457)

[Главное в задаче детекции 16](#_Toc136509458)

[Задача сегментации изображений 17](#_Toc136509459)

[Метрики качества сегментации 17](#_Toc136509460)

[Алгоритм сегментации объектов 18](#_Toc136509461)

[Функция потерь в задаче сегментации 18](#_Toc136509462)

[Выводы 18](#_Toc136509463)

[Полносвязные сети 19](#_Toc136509464)

[Почему нейросети 19](#_Toc136509465)

[Объекты и признаки 19](#_Toc136509466)

[Чем ещё полезны нейросети 20](#_Toc136509467)

[Библиотека Keras 22](#_Toc136509468)

[Задачи 22](#_Toc136509469)

[Логистическая регрессия 25](#_Toc136509470)

[Логистическая регрессия в Keras 30](#_Toc136509471)

[Задачи: 30](#_Toc136509472)

[Полносвязные нейронные сети 33](#_Toc136509473)

[Полносвязные нейронные сети в Keras 36](#_Toc136509474)

[Работа с изображениями в Python 38](#_Toc136509475)

[Задачи 38](#_Toc136509476)

[Цветные изображения 39](#_Toc136509477)

[Задачи 40](#_Toc136509478)

[Многоклассовая классификация 41](#_Toc136509479)

[Классификация изображений 47](#_Toc136509480)

[Задачи 47](#_Toc136509481)

[Обучение многослойной сети [GPU] 48](#_Toc136509482)

# Компьютерное зрение

Добро пожаловать на курс по компьютерному зрению! Здесь вы научитесь решать задачи классификации и регрессии для изображений.

## Структура курса

**Компьютерное зрение** (англ. computer vision) — область искусственного интеллекта, которая изучает способность машин видеть как человек. На этом курсе вы будете обучать машины понимать изображения или видео. Например, определять, какой фрукт на картинке или сколько лет человеку на фото.

Сегодня термин «компьютерное зрение» несколько устарел, и на смену ему пришёл новый — «обработка изображений». Компьютерным зрением теперь называют зрение роботов. Однако в рамках этого курса оба термина равнозначны.

С задачами компьютерного зрения успешно справляются нейронные сети. Работе с ними и будет посвящён курс.

Изучение нейронных сетей вы начнёте с устройства библиотеки Keras. От обучения простых сетей перейдёте к глубоким нейронным сетям. В конце курса вас ждёт самостоятельный проект: с нуля решите задачу обучения с учителем для изображений.

Нейронные сети могут обучаться долго, для комфортной практики мы разработали новый тренажёр. Все вычисления будут производиться на сервере с графической картой (GPU) [Yandex Compute Cloud](https://cloud.yandex.ru/services/compute).

## Ваши цели

* Изучить основы работы с изображениями.
* Познакомиться с библиотекой Keras.
* Разобраться, что такое полносвязные и свёрточные сети.
* Научиться обучать модель ResNet.

Запаситесь терпением: вам предстоит освоить много теории и инструментов работы с нейронными сетями. Вперёд!

### Проектная работа

#### Ознакомьтесь с проектом, который нужно выполнить в конце курса.

#### Описание проекта

Сетевой супермаркет «Хлеб-Соль» внедряет систему компьютерного зрения для обработки фотографий покупателей. Фотофиксация в прикассовой зоне поможет определять возраст клиентов, чтобы:

* анализировать покупки и предлагать товары, которые могут заинтересовать покупателей этой возрастной группы;
* контролировать добросовестность кассиров при продаже алкоголя.

Постройте модель, которая по фотографии определит приблизительный возраст человека. В вашем распоряжении набор фотографий людей с указанием возраста.

#### Инструкция по выполнению проекта

1. Проведите исследовательский анализ набора фотографий.
2. Подготовьте данные к обучению.
3. Обучите нейронную сеть и рассчитайте её качество.

#### Описание данных

Данные взяты с сайта [*ChaLearn Looking at People*](http://chalearnlap.cvc.uab.es/dataset/26/description/). Они находятся в папке /datasets/faces/.

В вашем распоряжении одна папка со всеми изображениями (/final\_files) и CSV-файл labels.csv с двумя колонками: file\_name и real\_age.

Извлечь данные из папки вам поможет новый метод [*ImageDataGenerator*](https://keras.io/preprocessing/image/) —flow\_from\_dataframe(dataframe, directory, ...).

#### Как будут проверять проект?

Мы подготовили критерии оценки проекта, которыми руководствуются ревьюеры. Прежде чем приступить к решению кейса, внимательно их изучите.

На что обращают внимание ревьюеры, проверяя проект:

* Все ли шаги инструкции выполняете?
* Как проводите исследовательский анализ данных?
* Как готовите данные к обучению?
* Какие нейронные сети рассматриваете?
* Какие выводы делаете?
* Следите ли за структурой проекта?
* Поддерживаете аккуратность кода?

Всё, что вам нужно знать, есть в шпаргалках и конспектах прошлых тем.

Успеха!

### Введение

В этой теме вы освоите базовые понятия компьютерного зрения: узнаете, что такое изображение, из чего оно состоит и как его можно преобразовывать. Вы также познакомитесь с разными типами задач компьютерного зрения — от классификации до сегментации — и выясните, для чего они применяются.

#### Что вы изучите?

* Чем различаются растровые и векторные изображения.
* Как устроена цветовая модель RGB.
* Какие параметры описывают изображение.
* Зачем нужны аугментации и как их совершить.
* Какие задачи решает компьютерное зрение.

### Что такое изображение

Скоро вам предстоит много работать с изображениями, поэтому необходимо детально разобраться с тем, что они из себя представляют.

## Типы изображений

На этом курсе вам предстоит работать с растровыми RGB-изображениями.

### Что значит «растровое»?

Цифровые изображения делятся на два типа: растровые и векторные. Первые состоят из пикселей, а вторые — из геометрических примитивов.

Почти все изображения, что вы видите в интернете, а также все фотографии — растровые. С векторной же графикой вы сталкиваетесь, когда читаете тексты с экрана: каждая буква на них — это векторное изображение.

Это связано с тем, что векторные изображения можно увеличить сколь угодно без потери качества, тогда как растровые от этого распадаются на «квадратики».

Главный недостаток векторной графики — она не позволяет создать содержательное изображение. Она удобна, чтобы создать логотип или эмблему, но нарисовать с её помощью реалистичный пейзаж практически невозможно.

### Что такое RGB?

Так называется самая распространённая цветовая модель, то есть способ математически описать какой-либо цвет. Её название — аббревиатура от английских слов red, green и blue — красный, зелёный и синий. Это основные цвета модели: если их смешать в определённой пропорции, можно получить почти любой другой цвет.

Каждый основной цвет отображается через свой канал. Соответственно, в RGB их три — под красный, зелёный и синий цвета. Каждый канал показывает свой цвет с определённой «силой».

Цвета в RGB смешиваются так же, как и цветные световые лучи. Модели, в основе которых такая идея, называются аддитивными (от англ. to add — «добавлять»). Представьте три фонарика: красный, зелёный и синий. Если включить их в тёмной комнате и направить на белую стену так, чтобы их свет пересекался, то в местах наложения световых лучей появятся другие цвета. Например, когда синий и красный лучи пересекаются, получается пурпурный. А жёлтый свет видно на пересечении зелёного и красного света. Смесь всех трёх цветов даёт белый, а места, куда свет фонариков не попадает, будут чёрными.

RGB используют на дисплеях и мониторах, в которых работу каждого канала обеспечивает «лампочка» — светодиод, электронная пушка или иной источник света. Так воспроизводится почти весь спектр видимых цветов и на старом телевизоре, и на вашем смартфоне.

Цвет в RGB задаётся тройкой чисел, каждое из которых принимает значение от 0 до 255, — например, [127, 53, 65]. Они обозначают интенсивность красного, зелёного и синего цветов — именно в таком порядке. Значит, чёрному цвету соответствует набор чисел [0, 0, 0], белому — [255, 255, 255], красному — [255, 0, 0], а пурпурному — [255, 0, 255]. Иногда числа записываются тремя числами без пробелов в шестнадцатеричном формате. Например, значению 255 соответствует FF, тогда красный цвет будет записан как FF0000 (255 00 00). Такое обозначение часто используется в графических редакторах.

## Что такое пиксели?

Пиксель — это мельчайшая неделимая частица растрового изображения. Цвет пикселя задаётся числовыми параметрами цветовой модели. В случае RGB их три, по числу каналов. Но при этом весь пиксель заполнен одним цветом: нельзя сказать, что одна половина пурпурного пикселя красная, а другая — синяя.

Одно из свойств изображения — разрешение. Оно описывает размер изображения, а именно его ширину и высоту в пикселях. Например, распространённый размер 4K предполагает разрешение 3840x2160. Это значит, что фото или видео такого размера содержит 3840 пикселей по горизонтали и 2160 по вертикали. Если умножить 3840 на 2160, то получится число пикселей на изображении — 8 294 400.

Изображения также характеризуются отношением ширины к высоте, то есть соотношением сторон. В случае 4K оно составляет 16:9, поскольку 38402160=16921603840​=916​.

Самое главное, что нужно запомнить: цвет каждого пикселя определяется тремя значениями каналов, но при этом пиксель «покрашен» только одним цветом.

## Параметры изображения

Три главных параметра растрового RGB-изображения — ширина, высота и количество каналов. Далее мы будем сталкиваться с другим числом каналов, обычно все вместе они записываются так: 3840x2160x3 — ширина 3840 пикселей, высота 2160 пикселей и три канала.

## Как посчитать размер изображения

Вы уже знаете, что цвет каждого пикселя на растровом RGB-изображении определяется тремя числами. Однако это не всё: они же определяют объём памяти, который занимает пиксель. Каждое из этих чисел может принимать 256 значений — от 0 до 255 включительно. Это значит, что для работы с RGB-изображением используется беззнаковый целочисленный восьмибитовый тип данных (unsigned int8): 8 бит будет достаточно, чтобы хранить каждое число.

Чтобы посчитать размер изображения, нужно перемножить ширину, высоту и число каналов. В результате получается количество чисел, определяющих размер изображения. Каждое из них весит 8 бит, или 1 байт. Это значит, что с помощью перемножения получается вес картинки в байтах. Например, столько будет занимать 4К-фотография:

3840⋅2160⋅3=248832003840⋅2160⋅3=24883200

Это примерно 24 мегабайта — очень много для одной не самой детальной фотографии. Помните 100-мегапиксельные камеры из рекламы? А тут их всего 8!

Здесь помогает сжатие файлов. Для этого их хранят в специальных форматах, два самых популярных — JPEG и PNG. Первый сжимает сильнее и с потерей качества картинки, второй — слабее и без потерь. Именно поэтому ваши фотографии не забивают всю память через пару дней.

Когда изображение открывается во время обучения, оно оказывается в оперативной памяти в несжатом виде. Так что ваше ОЗУ может испариться в мгновение ока, если вы потеряете бдительность. 1000 открытых 4К-картинок — это почти 17 гигабайт оперативной памяти. При переносе изображений на память видеокарты об этом тоже стоит помнить, там обычно памяти ещё меньше.

Посчитайте, сколько памяти в байтах будет занимать цветное изображение с разрешением 1280х720.

Ваш ответ правильный2764800

Правильный ответ: 2764800

## Выводы

На этом курсе вы будете работать с растровыми RGB-изображениями. Они состоят из неделимых (по цвету) пикселей. Главные параметры изображения: ширина и высота в пикселях, а также количество каналов. Размер несжатого изображения в байтах — произведение высоты, ширины и количество каналов.

### Аугментации изображений

Аугментации — это увеличение, «раздутие» выборки данных, при котором новые данные не собираются, а автоматически создаются из уже существующих. Эта техника применяется и к изображениям, чтобы повысить качество работы модели, которая на них обучается.

Прежде чем перейти к аугментациям, ещё раз посмотрите, что представляют собой изображения. К примеру, так в числовом виде выглядит красный канал картинки размером 2х2 пикселя:

Скопировать кодPYTHON

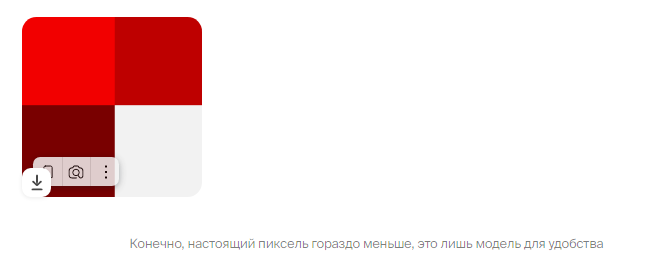
[

[255, 200],

[127, 0]

]

Изображения хранятся как массив в массиве, обычно это объекты типа numpy.ndarray. Поэтому большинство преобразований изображений — преобразования NumPy-массивов. Если сильно увеличить картинку, то она будет выглядеть так:

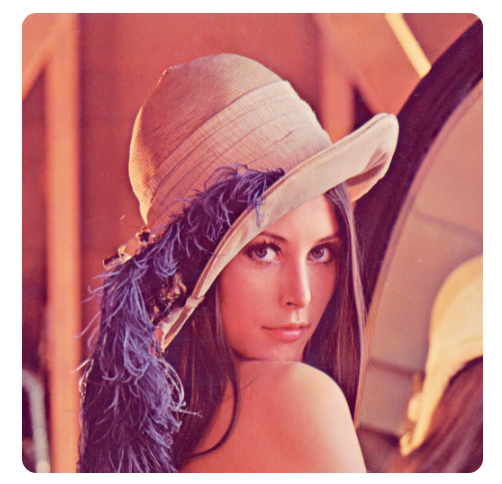


### OpenCV

OpenCV (англ. Open Computer Vision) — библиотека, разработка которой началась ещё в 1999 году. С тех пор она обросла самыми разными алгоритмами. OpenCV особенно широко использовалась до роста популярности нейросетей, но развивается и сегодня. Большинство её алгоритмов не устарело, а нашло нишевое применение. Такие методы называют классическим компьютерным зрением — в противовес нейросетевому. Чтобы открыть изображение в OpenCV, используйте метод imread:

import cv2

image = cv2.imread('Lenna.png')



💡 Кстати, знакомьтесь — это Лена, стандартная картинка, которую используют, чтобы продемонстрировать алгоритмы обработки изображений. На ней запечатлена Лена Форсен, шведская модель, чей портрет случайно стал знаковым. В 1973 году Александр Савчук из Университета Южной Калифорнии искал подходящее фото, чтобы проиллюстрировать свою научную статью. Под рукой оказался журнал Playboy — а в нём был постер с Леной.

Перед вами изображение типа numpy.ndarray. С помощью image.shape можно узнать его размер — (512, 512, 3), это 512x512 пикселей и три канала. OpenCV открывает изображение в цветовой модели BGR — RGB с другим порядком каналов. Чтобы вернуть его в стандартный вид, достаточно развернуть порядок классическим для Питона способом: image[...,::-1].

### Отражение по горизонтали и вертикали

Самые простые преобразования — отражения по горизонтали и вертикали, или флипы (от англ. to flip — «переворачивать»). Для отражения по горизонтали нужно перевернуть строки изображения, а для отражения по вертикали — столбцы:

Скопировать кодPYTHON

Исходное изображение:

[

[255, 200],

[127, 0]

]

Отражение по горизонтали:

[

[200, 255],

[0, 127]

]

Отражение по вертикали:

[

[127, 0],

[255, 200]

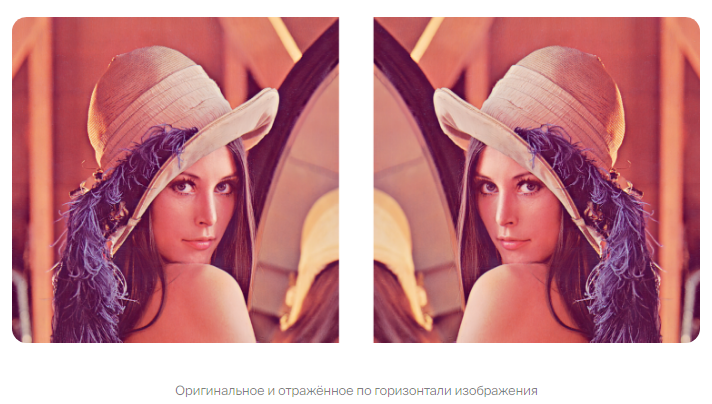
]

Или в виде изображений:



Отражение можно сделать разными способами:

* Встроенными средствами Питона   
  Оператор, которым вы развернули порядок каналов, может изменить и изображение. Достаточно выбрать нужное измерение и развернуть порядок строк. Отражение по горизонтали осуществляется так: image[:,::-1,:]. Если же поменять первое измерение, то произойдёт вертикальное отражение.
* NumPy   
  В NumPy есть специальный метод flip, который принимает массив и ось, по которой надо сделать флип. 0 — вертикальный флип, 1 — горизонтальный, 2 — в области каналов: например, np.flip(image, 1).
* OpenCV   
  В OpenCV идентичный NumPy синтаксис: cv2.flip(image, 1). Однако флип по каналам так сделать нельзя, в OpenCV cv2.cvtColor преобразует каналы и цветовые модели.



### Яркость, контраст и гамма

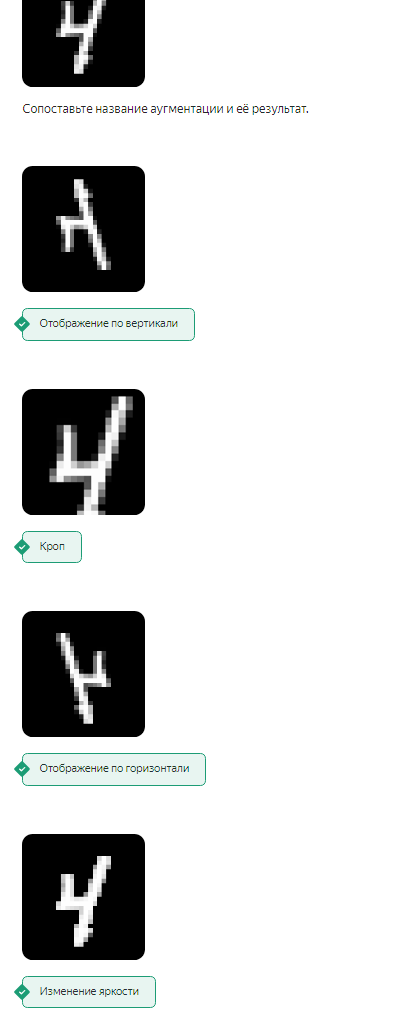
Чтобы увеличить яркость изображения, прибавьте к нему определённое число. Это значит, что нужно взять каждое значение канала по всем пикселям на картинке и увеличить его на X. Остальные операции выполняются аналогично: увеличить или уменьшить контраст — умножить на число, а изменить гамму — возвести в степень. Эти операции выполняются очень просто: например, image+10 увеличит яркость, а image-10 — уменьшит.



Важные замечания:

* В современных редакторах яркость, контраст и гамма изменяются уже не так просто. Алгоритмы усложняют, чтобы сделать изображения визуально привлекательными. Для большинства задач машинного обучения всё ещё достаточно базовых операций.
* Если вы будете использовать в коде только базовую команду image+10, то преобразование может получиться неестественным. Это происходит из-за переполнения беззнакового целочисленного восьмибитового типа данных. Он хранит только 256 чисел, и если прибавить 10 к значению 247, то данные не будут отображаться корректно. Это можно исправить, если перезаписать значение в 32-битовом формате, прибавить 10, а затем вернуться к восьми битам: np.clip((image.astype(np.int32) + 10), 0, 255).astype(np.uint8). Ограничения типов данных и их преобразования — отдельная захватывающая тема.
* Чтобы изменить контраст и гамму, изображение нужно «отцентрировать», то есть перевести диапазон значений каналов из [0, 255] в [-1, 1]. При этом не забывайте о типе данных, который вы используете.

К счастью, на практике достаточно только понимать, как осуществляются преобразования. Делать вручную их не приходится — за вас их совершат различные библиотеки, с одной из которых вы познакомитесь позднее.



## Выводы

Аугментации используются, чтобы улучшить обучение. Все преобразования изображений — преобразования NumPy-массивов. В большинстве случаев для аугментации хватает готовых библиотек, которые вы изучите позднее, но иногда придётся создавать методы с помощью OpenCV и других библиотек.

## Задача классификации изображений

Компьютерное зрение помогает решать разные задачи, их тип определяется целью, которой нужно достичь. Например, задача классификации изображений состоит в том, чтобы предсказать, к какому классу относится изображение. Пример такой задачи — научить модель отличать фото котов от снимков собак.



Задача классификации изображений очень похожа на задачу классификации табличных данных, с которой вы знакомы, но у неё есть некоторые особенности.

## Ограничения классов

Предположим, вы превосходно обучили модель, и теперь она ни за что не перепутает кошку с собакой. Но что будет, если подать на вход изображение попугая или дома? Модель не может выдать какой-либо ответ кроме «кот» или «собака», поэтому она случайным образом отнесёт такую картинку к какому-то из двух классов. Запомните: модель корректно работает только с изображениями тех классов, которые она обучалась предсказывать. Предсказания, сделанные на других изображениях, нельзя воспринимать всерьёз.

## Препроцессинг данных

Перед классификацией изображений проводится их препроцессинг, или предобработка. Процедура предполагает как минимум два этапа: изменение размера изображений до подходящего для сети и нормализация изображений. Именно на этом этапе происходят аугментации — повороты изображений, изменения яркости, насыщености, случайный кроп или добавление шума. Препроцессинг данных сильно меняется от задачи к задаче.

## Модели компьютерного зрения

Из-за ряда ограничений классические алгоритмы машинного обучения, которые вы изучали ранее, недостаточно эффективны в задачах классификации изображений. Поэтому современное компьютерное зрение основано на нейронных сетях. В следующих темах вы подробно разберёте, что это такое и как они работают.

## Функции потерь и метрики качества

Для любых задач классификации функции потерь одинаковы: бинарная кросс-энтропия для двухклассовой классификации и обычная кросс-энтропия для случая, где классов больше двух. Классификация изображений — не исключение.

Метрики качества также не сильно отличаются и зависят от задачи: точность (accuracy), f1-метрика, ROC-AUC и т. д.

## Обучение модели классификации изображений

Порядок обучения модели классификации картинок мало отличается от работы с табличными данными:

1. Загрузка изображений и их классов из датасета;
2. Предобработка изображений, аугментации;
3. Нейросеть обучается.

## Выводы

Классификация изображений практически не отличается от классификации табличных данных. Главные различия:

* Изображения — объёмные сложные данные, для их хранения и предобработки требуются особые методы.
* Для классификации изображений используются нейросети, с которыми вы познакомитесь совсем скоро.

В следующем уроке мы рассмотрим более комплексные задачи компьютерного зрения. Но на практике вы освоите именно классификацию — поэтому важно понимать и учитывать эти особенности.

## Задачи детекции и сегментации

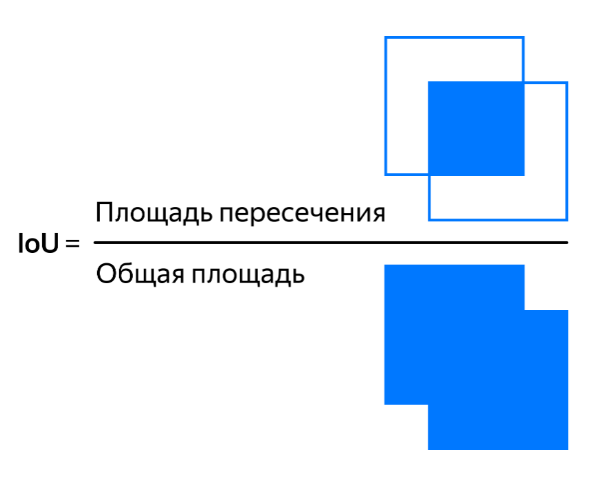
Свёрточные нейросети справляются не только с классификацией изображений, но и с другими задачами.

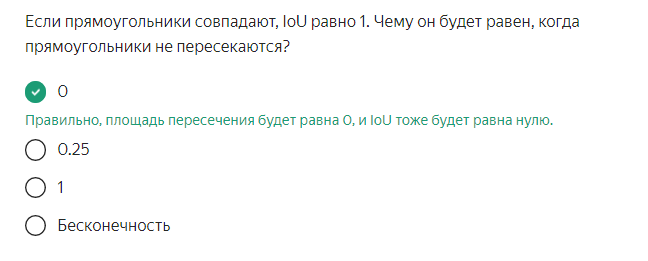
### Задача детекции объектов

Задача детекции состоит в том, чтобы найти на изображении объекты и выделить их прямоугольником. Она бывает двух типов: обнаружение объектов одного класса и объектов разных классов. Задачу детекции также называют задачей локализации объектов.

## Метрики качества детекции

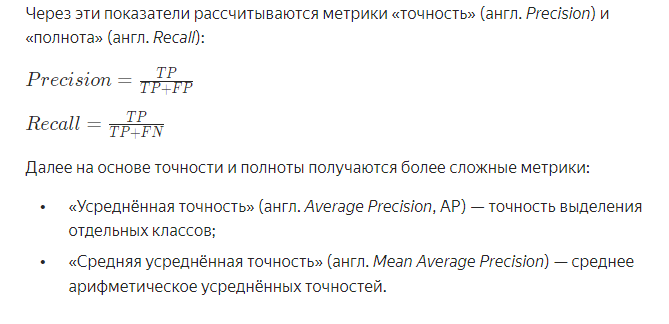
Большинство метрик детекции объектов учитывают показатель IoU (англ. Intersection over Union): это отношение площади пересечения двух прямоугольников к их общей площади:





С помощью IoU вычисляют следующие показатели:

* **True Positive (TP)** — объект корректно выделен прямоугольником. Полученная моделью область выделения может не совпадать с описанием в валидационном датасете. Поэтому выделение считается верным, если IoU больше заданного порогового значения (англ. *threshold*): IoU ≥ threshold.
* **False Positive (FP)** — объект некорректно выделен прямоугольником, IoU < threshold.
* **False Negative (FN)** — модель вообще не выделила объект;
* **True Negative (TN)** — модель корректно НЕ выделила фоновую область (ту, где нет объектов). Этот показатель практически никогда не используется.



## Алгоритм детекции объектов

Большинство подходов к детекции объектов сводится к двум алгоритмам действий.

* Двухэтапный, задача разбивается на два шага:  
  1. Выделить прямоугольные рамки разных размеров, которые предположительно содержат объект — области интереса (англ. regions of interest).
  2. Каждая из областей интереса подаётся на вход свёрточной нейронной сети, которая классифицирует объект и пытается «подогнать рамки», чтобы уточнить выделение объекта.

Двухэтапный алгоритм требует много времени на предсказания, но выделяет объекты более качественно.

* Одноэтапный, при котором ради более высокой скорости работы опускается поиск областей интереса. Но за эту скорость приходится платить качеством модели.   
  Одна из самых популярных одноэтапных моделей — YOLO (англ. You Look Only Once). Алгоритм предсказания YOLO:   
  1. Всё изображение делится на мелкие части;
  2. Модель предсказывает, к какому классу относится каждая часть;
  3. Части одного объекта объединяются в большие объекты;
  4. С помощью алгоритма Non-Maximum Suppression (NMS) убираются избыточные предсказания.



## Функция потерь в задаче детекции

Для задачи детекции нужно правильно предсказать и класс, и границы прямоугольника, поэтому функция потерь состоит из нескольких частей:

* Обычная кросс-энтропия для классификации частей изображения;
* Функция потерь локализации, которая учитывает ошибки в координатах;
* Функцию потерь на классификацию всего найденного прямоугольника.

## Главное в задаче детекции

Если в ближайшем будущем вам понадобиться сделать свой детектор — обратите внимание на различные варианты YOLO. Многие из них уже готовы к использованию, достаточно правильно подготовить данные и запустить нужную команду в консоли.

# Задача сегментации изображений

В задаче классификации нужно предсказать класс всего изображения, в задаче детекции — каждого объекта. А в задаче сегментации — каждого пикселя. Результат предсказания — ещё одно изображение, которое может выглядеть так:



Каждый цвет — отдельный класс: кресло, стена, потолок и т. д.

Задача сегментации делится на подзадачи. Если вам нужно присвоить каждому пикселю определённый класс, то перед вами задача семантической сегментации (англ. semantic segmentation). На выходе модель семантической сегментации должна вернуть маску — изображение, в котором объекты одного класса должны иметь один и тот же цвет. При этом, если на фото присутствуют два или три дивана, то они будут иметь один класс.

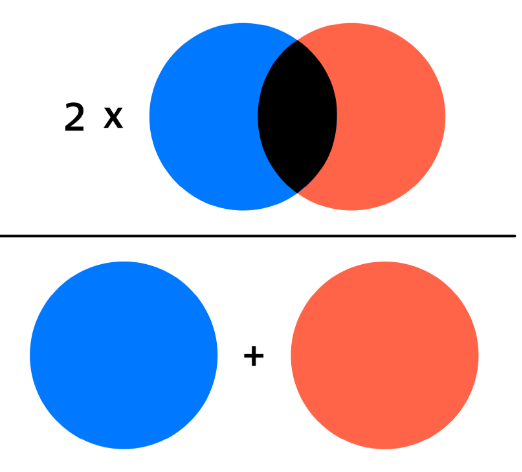
В задаче сегментации экземпляров (англ. instance segmentation) каждому пикселю присваивается определённый экземпляр класса. В таком случае два дивана будут считаться разными объектами. На выходе возвращается список объектов с описанием их местонахождения, присвоенным классом и точностью предсказания. Местоположение объекта описывается координатами контура или маской — бинарной матрицей размером с изображения на входе. 0 обозначает отсутствие объекта, 1 — присутствие.

## Метрики качества сегментации

В задачах семантической сегментации каждому пикселю присваивается определённый класс, что позволяет посчитать попиксельную долю правильных ответов (англ. Pixel Accuracy). Она равна доле пикселей, для которой модель правильно присвоила класс.

Для семантической сегментации также подходит метрика IoU. Вместо прямоугольников учитываются фигуры, ограниченные контурами.

Помимо IoU можно использовать коэффициент Сёренсена-Дайса, или коэффициент Дайса, (англ. Dice coefficient) — отношение удвоенной площади пересечения к сумме площадей предсказания и реального объекта.



В задаче сегментации экземпляров используются те же самые метрики, что и для детекции объектов, однако учитывается не суммарная площадь объектов, а площадь каждого из них.

## Алгоритм сегментации объектов

Классическая модель для семантической сегментации — [U-Net](https://arxiv.org/abs/1505.04597).

Как и многие другие модели компьютерного зрения, U-Net — это нейронная сеть. Она была разработана, чтобы находить патологии на медицинских снимках. U-Net «смотрит» на картинку в разных масштабах и определяет, к какому классу относится каждый из пикселей.

## Функция потерь в задаче сегментации

Функция потерь в задаче семантической сегментации в простом случае — кросс-энтропия на пиксельном уровне. Для сегментации экземпляров используются более сложные функции потерь.

## Выводы

Задачи детекции — это поиск объектов и их выделение. Ключевая метрика — Intersection over Union. Архитектура YOLO — наиболее доступный вариант, чтобы научиться работать с задачами детекции.

Задача сегментации — это определение класса каждого пикселя в изображения. В процессе модель создаёт маску сегментации. Основная архитектура для задачи сегментации — U-Net.

Чтобы разобраться в архитектурах вроде YOLO или U-Net, необходимо изучить основы нейронных сетей, чему и посвящены следующие темы.

## Полносвязные сети

# Почему нейросети

Нейросети справляются с задачами компьютерного зрения лучше, чем модели других типов — пора разобраться, почему.

## Объекты и признаки

Линейная регрессия, случайный лес и градиентный бустинг работают хорошо на табличных данных: в строках записываются объекты, а в столбцах — их признаки. А [есть](https://www.facelytics.io/) датасеты из фотографий, по которым нужно определить пол или возраст человека. Что тогда будет объектом и признаком?

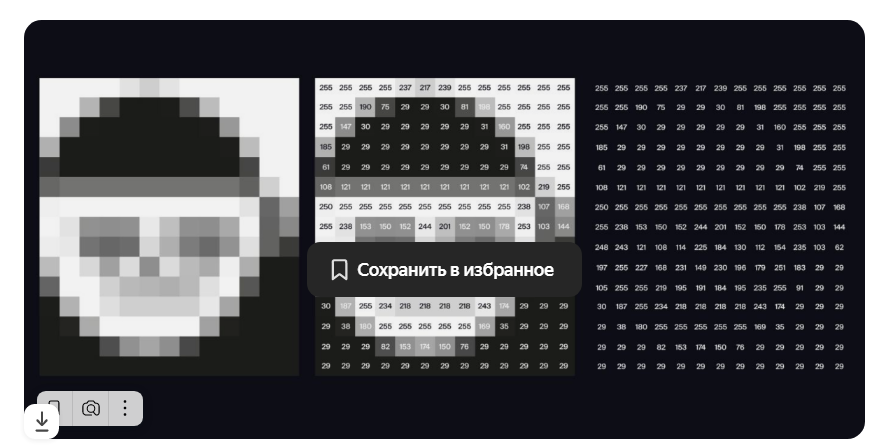
Предположим, что объект — это фотография, а каждый пиксель — признак этого объекта.

**Пример**

Вы уже знаете, что алгоритм «видит» пиксель как набор чисел, характеризующих его цвет. В случае с цветными RGB-изображениями каждый пиксель описан набором из трёх чисел, которые обозначают интенсивность красного, зелёного и синего цветов. В большинстве случаев мы сталкиваемся именно с такими картинками.

Однако для простоты вы начнёте осваивать компьютерное зрение с одноцветных, чёрно-белых изображений. Цвет каждого их пикселя — оттенок серого, заданный одним числом.

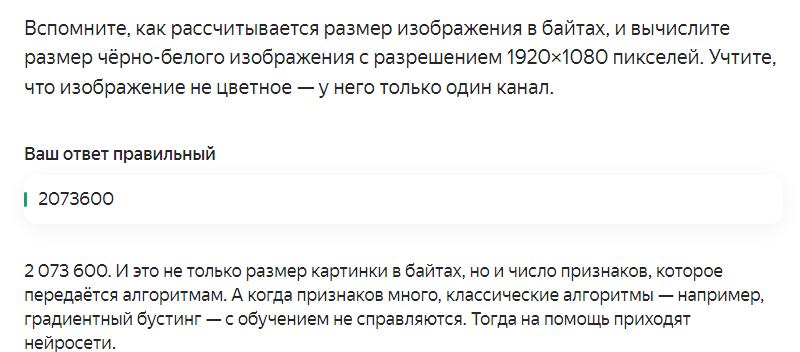
На первой картинке чёрно-белое изображение низкого разрешения. На второй — это же изображение с указанием оттенков серого цвета от 0 (чёрный) до 255 (белый). Именно в таком диапазоне лежит значение яркости пикселя чёрно-белого изображения, то есть возможное значение признака. А на третьей — сами эти цифры.



Чтобы получить признаки, нужно представить значения пикселей в виде вектора:

Скопировать код

[255, 255, 255, 255, 237, 217, 239, 255, 255, 255, 255, 255, 255, 255, 255, 190, 75, 29,



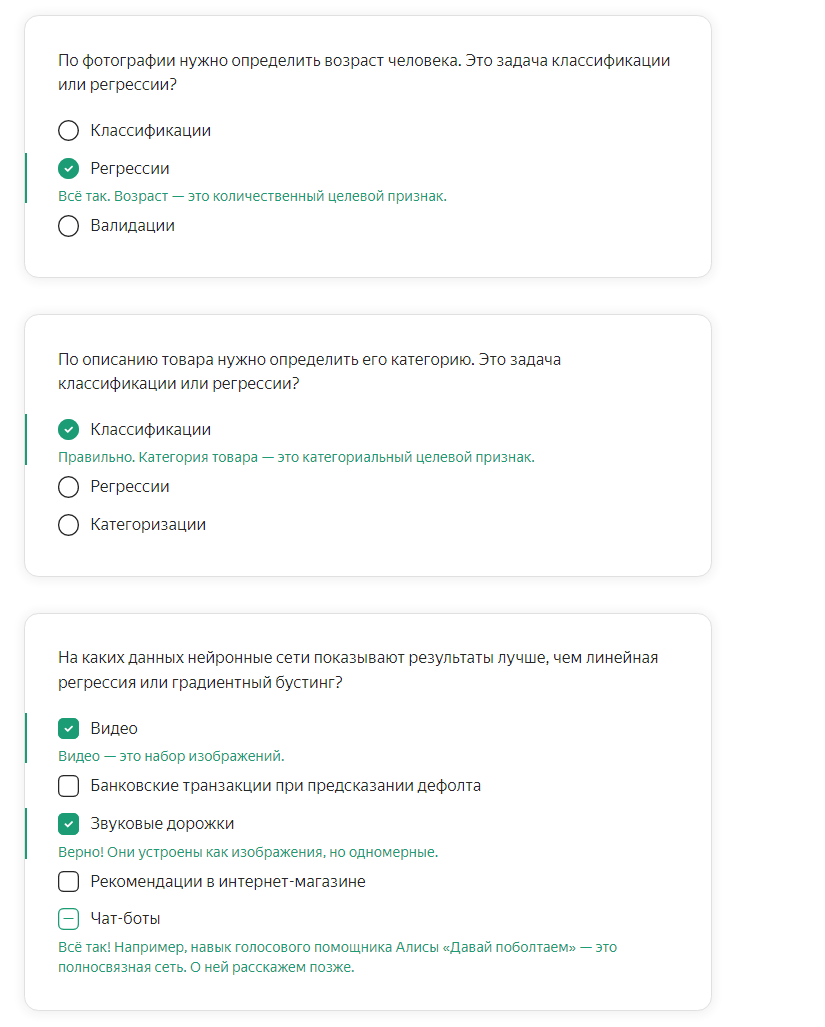
### Чем ещё полезны нейросети

Рассмотрите другой пример. Датасет — текстовые описания товаров в интернет-магазине. Нужно определить категорию, к которой принадлежит каждый товар.

Вы помните, что преобразовать текст в численный формат поможет техника мешка слов. Сначала для каждого слова создаётся новый столбец. Если слово есть в описании товара, в столбце ставится 1, если нет — 0. Если в описании 10–20 слов, логистическую регрессию на преобразованном датасете можно обучить. Когда слов сотни, этот способ кодирования уже не работает. Но избавившись от порядка следования слов, можно потерять и смысл текста. Если же применить n-граммы, признаков станет ещё больше: классические методы снова не справятся.

Что объединяет изображения и тексты?

1. Информация в них избыточна. Например, пиксели фона фотографии не так значимы при определении возраста, как морщины или седина. А предлоги и союзы не столь важны для определения смысла текста, как существительные и глаголы. Справиться с большим числом признаков помогут нейронные сети.
2. Соседние признаки связаны друг с другом. Зачастую соседние пиксели на изображении принадлежат одному объекту, а рядом стоящие слова в предложении связаны по смыслу. Если перемешать столбцы в табличных данных, их смысл не поменяется; если пиксели в изображении — будет шум, с которым уже ничего сделать нельзя. Учитывать порядок признаков умеют нейронные сети.



### Библиотека Keras

Keras (греч. κέρας, «рог») — это простая и компактная библиотека, созданная инженером компании Google Франсуа Шолле. По сути это интерфейс для работы с другой, более сложной библиотекой — TensorFlow (англ. «поток тензоров»). Это открытая библиотека, разработка команды исследовательского проекта Google Brain.

Сперва вы узнаете, как написать линейную регрессию в Keras. Линейная регрессия — это тоже нейронная сеть, но лишь с одним нейроном!

В начале темы вы будете работать с данными из прошлых курсов:

#### Задачи

1.

Обучите модель линейной регрессии средствами библиотеки Keras. Метод fit() напечатает на экране прогресс обучения и значение ошибки. Чтобы формат ответа был понятным, добавьте в этот метод аргумент verbose=2, где 2 означает вывод в консоль. Если указать 0, то его не будет вовсе; если — 1, то вывод предназначен для Jupyter Notebook.

import pandas as pd

from sklearn.linear\_model import LinearRegression

from tensorflow import keras

data = pd.read\_csv('/datasets/train\_data\_n.csv')

features = data.drop('target', axis=1)

target = data['target']

#model = LinearRegression()

#model.fit(features, target)

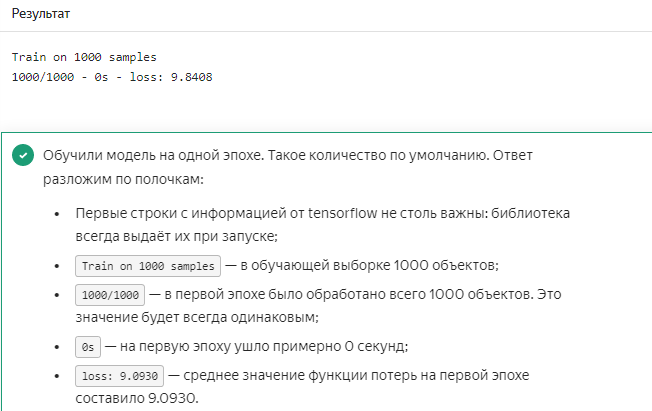
model = keras.models.Sequential()

keras.layers.Dense(units=1, input\_dim=features.shape[1])

model.add(keras.layers.Dense(units=1, input\_dim=features.shape[1]))

model.compile(loss='mean\_squared\_error', optimizer='sgd')

model.fit(features, target,verbose=2)



2.

Подберите такое количество эпох, чтобы получить MSE стабильно меньше 6.55. Добавить эпохи можно, указав аргумент epochs в методе model.fit().

import pandas as pd

from tensorflow import keras

data = pd.read\_csv('/datasets/train\_data\_n.csv')

features = data.drop('target', axis=1)

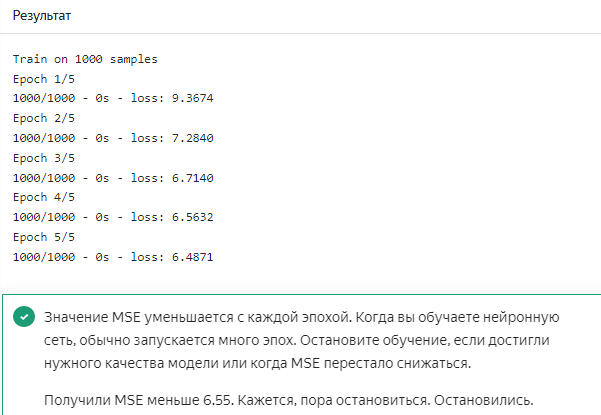
target = data['target']

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=1, input\_dim=features.shape[1]))

model.compile(loss='mean\_squared\_error', optimizer='sgd')

model.fit(features, target, verbose=2,epochs=5)



3.

Найдите значение функции потерь на валидационной выборке. Передайте модели валидационную выборку в аргументе validation\_data метода model.fit(). Загрузка валидационной выборки уже в прекоде.

import pandas as pd

from tensorflow import keras

data\_train = pd.read\_csv('/datasets/train\_data\_n.csv')

features\_train = data\_train.drop('target', axis=1)

target\_train = data\_train['target']

data\_valid = pd.read\_csv('/datasets/test\_data\_n.csv')

features\_valid = data\_valid.drop('target', axis=1)

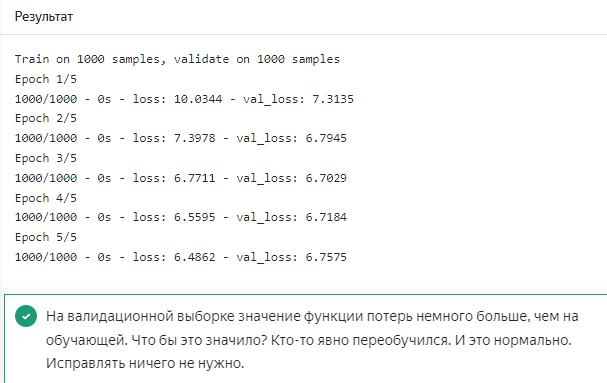
target\_valid = data\_valid['target']

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1]))

model.compile(loss='mean\_squared\_error', optimizer='sgd')

model.fit(features\_train, target\_train, verbose=2,epochs=5, validation\_data=(features\_valid,target\_valid))

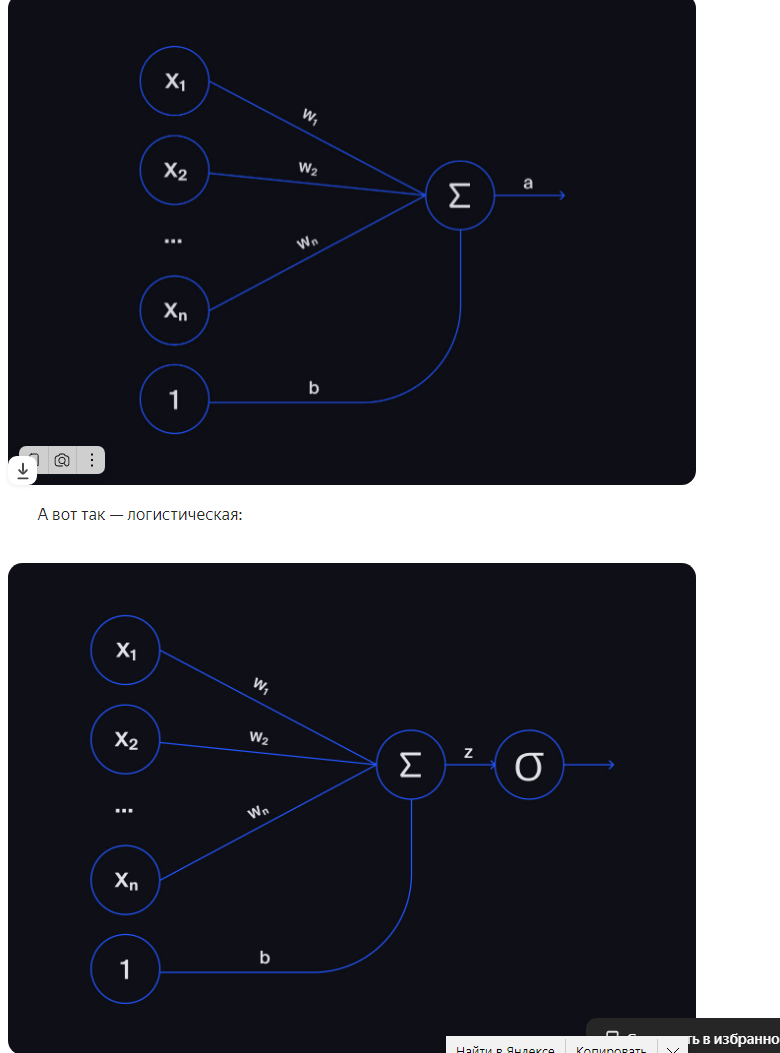


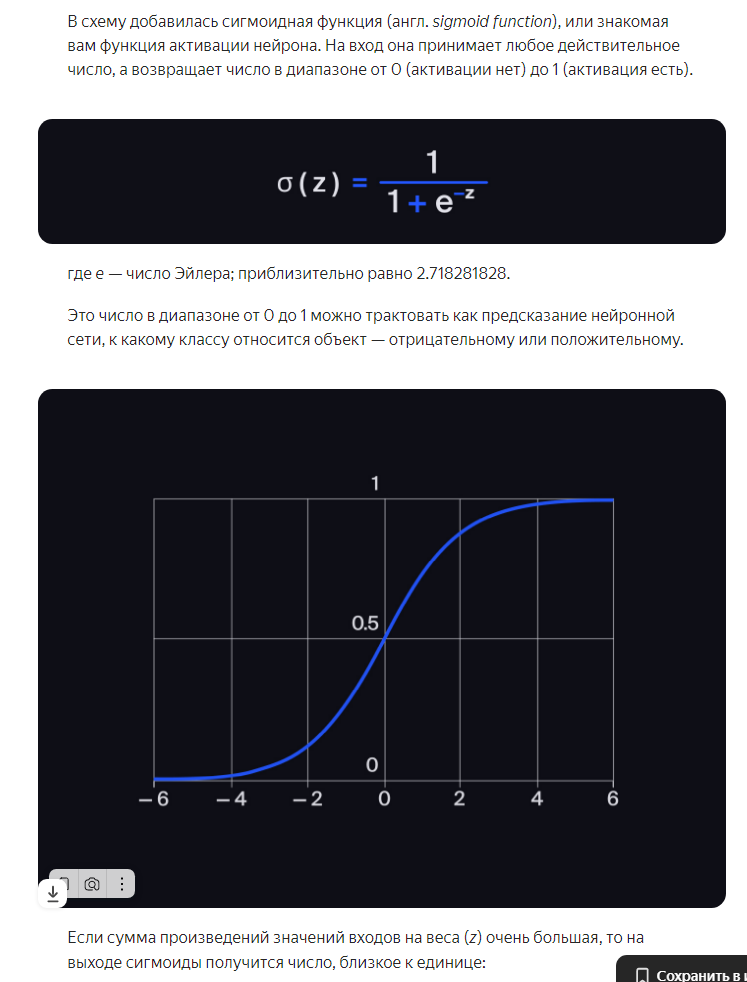
### Логистическая регрессия

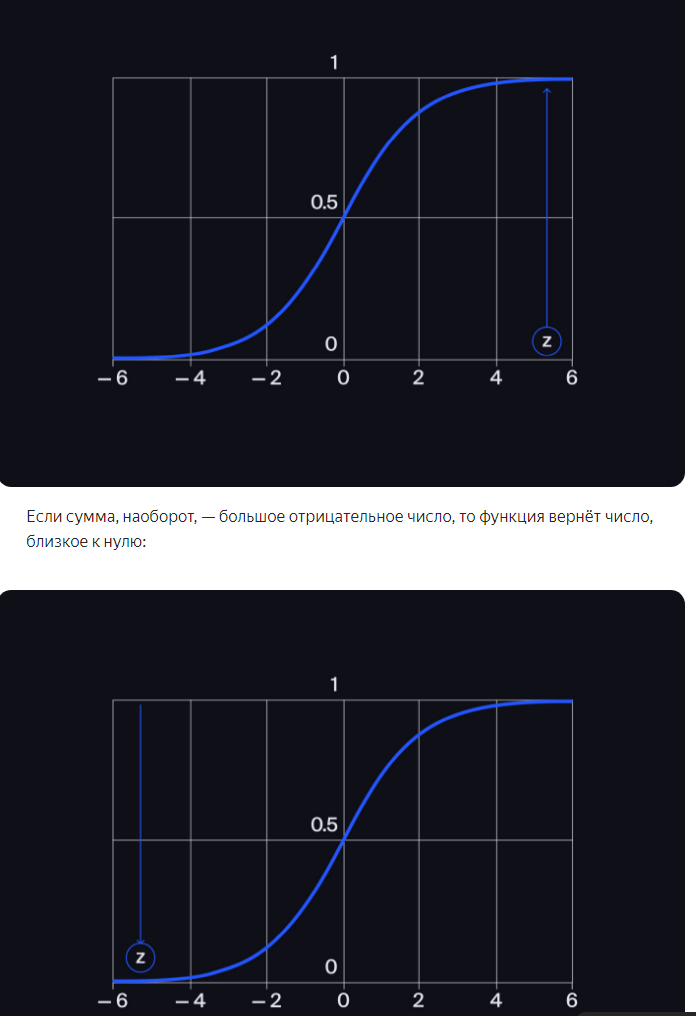
Линейная регрессия — это нейронная сеть с одним нейроном. Логистическая — тоже. Чем же они различаются?

Если классов у объектов всего два, то разница между линейной и логистической регрессиями почти незаметна. Нужно добавить всего один элемент.

Вот как выглядит линейная регрессия:

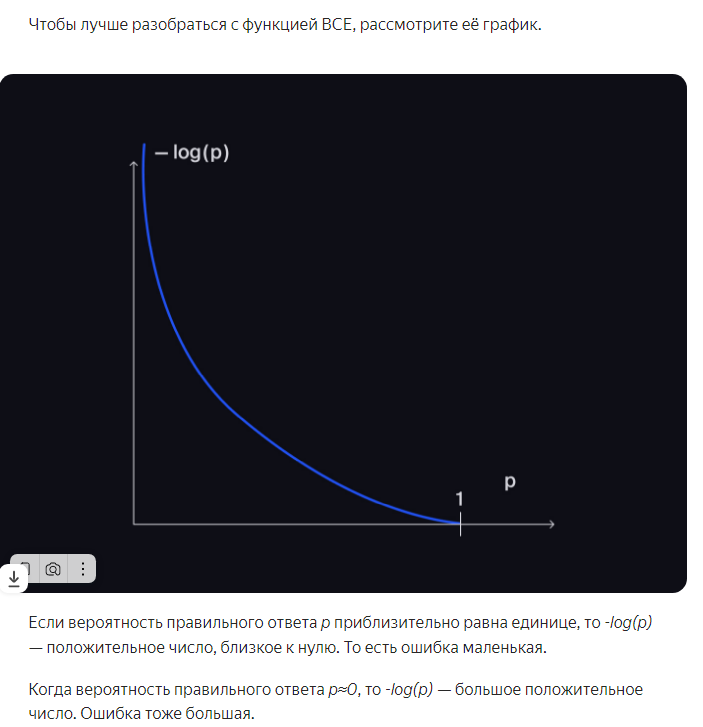
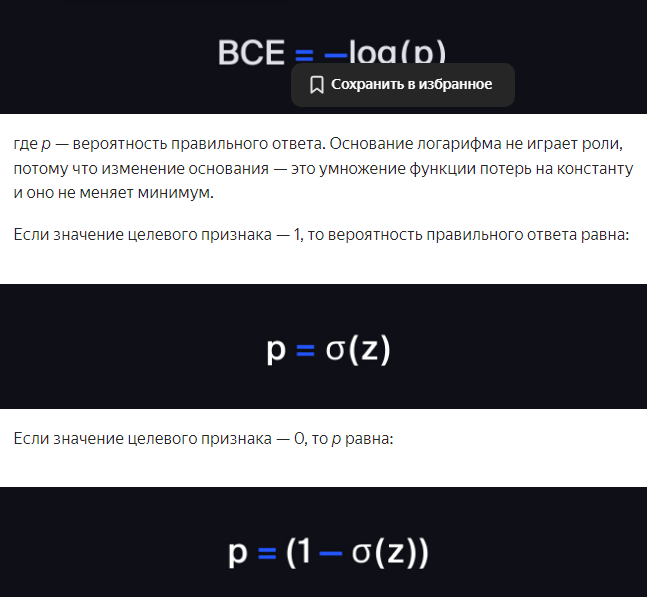


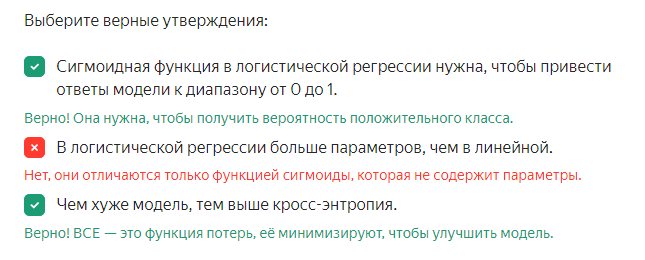




Функция потерь меняется в зависимости от типа нейронной сети. Если в задаче регрессии применяли MSE, то для бинарной классификации подходит Binary Cross-Entropy, BCE (англ. «бинарная кросс-энтропия»). Метрику accuracy применять не получится: у неё нет производной, поэтому SGD работать не будет.

ВСЕ вычисляется так:





### Логистическая регрессия в Keras

Вы написали линейную регрессию в Keras. Пора приступить к логистической.

Чтобы получить логистическую регрессию, код линейной нужно поменять лишь в двух местах:

1. К полносвязному слою примените функцию активации:

Скопировать кодPYTHON

keras.layers.Dense(units=1, input\_dim=features\_train.shape[1],

activation='sigmoid')

1. Поменяйте функцию потерь MSE на binary\_crossentropy.

Скопировать кодPYTHON

model.compile(loss='binary\_crossentropy', optimizer='sgd')

#### Задачи:

1.

Обучите логистическую регрессию на данных, загруженных в прекоде. Установите количество эпох, равным пяти.

Чтобы напечатать прогресс обучения, задайте аргумент verbose=2 в функции fit().

import pandas as pd

from tensorflow import keras

df = pd.read\_csv('/datasets/train\_data\_n.csv')

df['target'] = (df['target'] > df['target'].median()).astype(int)

features\_train = df.drop('target', axis=1)

target\_train = df['target']

df\_val = pd.read\_csv('/datasets/test\_data\_n.csv')

df\_val['target'] = (df\_val['target'] > df['target'].median()).astype(int)

features\_valid = df\_val.drop('target', axis=1)

target\_valid = df\_val['target']

# < напишите код здесь >

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1], activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='sgd')

model.fit(features\_train, target\_train, verbose=2,epochs=5, validation\_data=(features\_valid,target\_valid))



2.

Найдите accuracy модели на валидационной выборке. Предсказания модели вычисляются функцией predict(), как в sklearn. Сигмоида вернёт числа от 0 до 1, преобразуйте их в классы, сравнив с 0.5.

Напечатайте на экране значение accuracy (уже в прекоде). Чтобы выводу не мешал прогресс обучения, задайте verbose=0.

import pandas as pd

from tensorflow import keras

from sklearn.metrics import accuracy\_score

df = pd.read\_csv('/datasets/train\_data\_n.csv')

df['target'] = (df['target'] > df['target'].median()).astype(int)

features\_train = df.drop('target', axis=1)

target\_train = df['target']

df\_val = pd.read\_csv('/datasets/test\_data\_n.csv')

df\_val['target'] = (df\_val['target'] > df['target'].median()).astype(int)

features\_valid = df\_val.drop('target', axis=1)

target\_valid = df\_val['target']

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1],

activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='sgd')

model.fit(features\_train, target\_train, epochs=5, verbose=0,

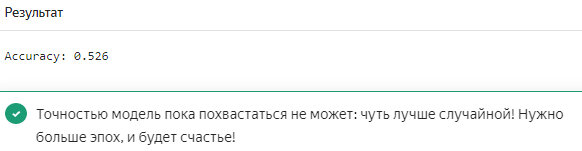
validation\_data=(features\_valid, target\_valid))

# < напишите код здесь >

predictions = model.predict(features\_valid) > 0.5

score = accuracy\_score(target\_valid,predictions )

print("Accuracy:", score)



3.

Обучение нейронной сети обычно занимает много времени. Сделайте так, чтобы отследить качество модели можно было на каждой эпохе. Для этого добавьте параметр metrics=['acc'] (от англ. *accuracy*) в методе compile().

Чтобы улучшить значение accuracy, обучите модель на десяти эпохах.

Подсказка

Обучите модель так:

Скопировать кодPYTHON

model.fit(features\_train, target\_train, epochs=10, verbose=2,

validation\_data=(features\_valid, target\_valid))

…

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1],

activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='sgd',metrics=['acc'])

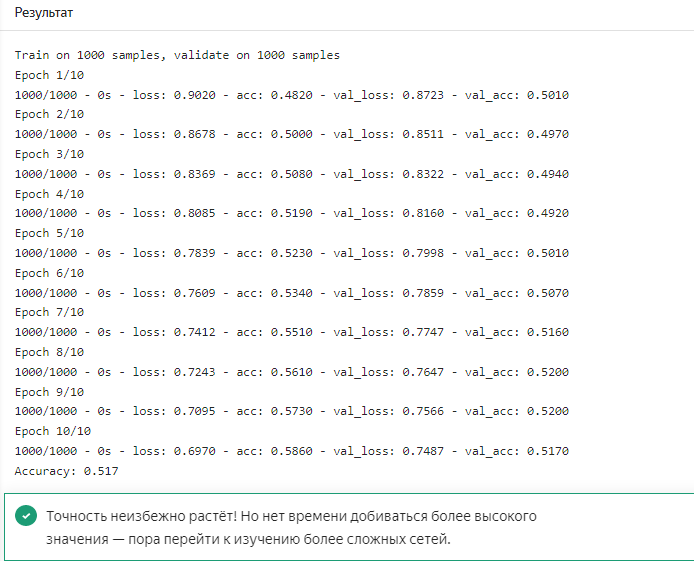
model.fit(features\_train, target\_train, epochs=10, verbose=2,

validation\_data=(features\_valid, target\_valid))

predictions = model.predict(features\_valid) > 0.5

score = accuracy\_score(target\_valid,predictions )

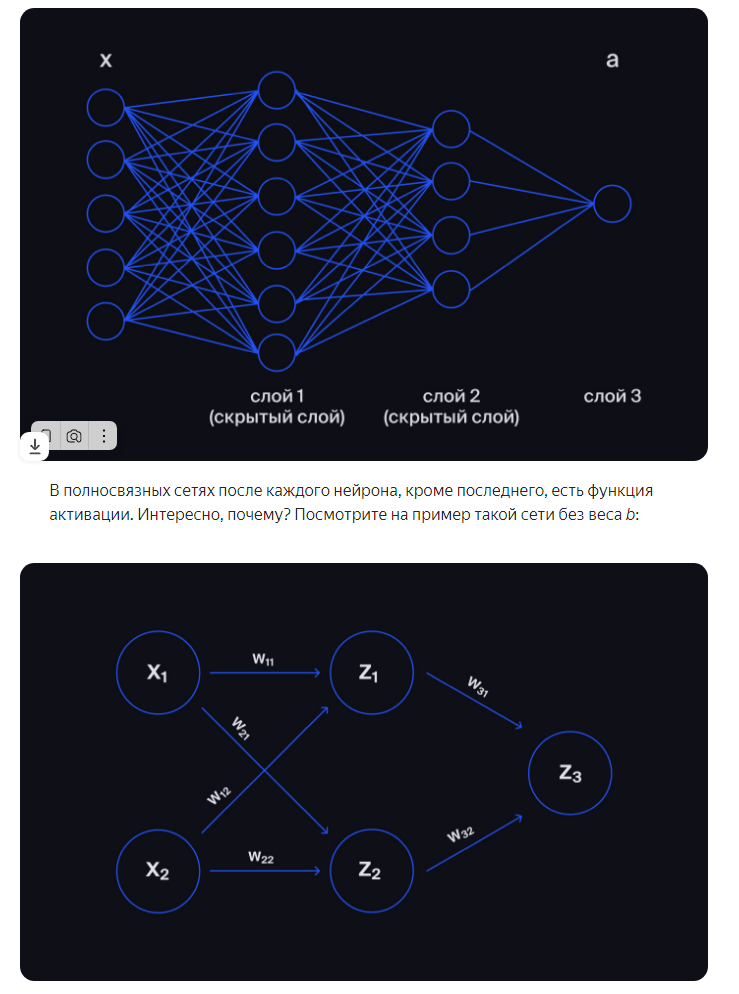
print("Accuracy:", score)

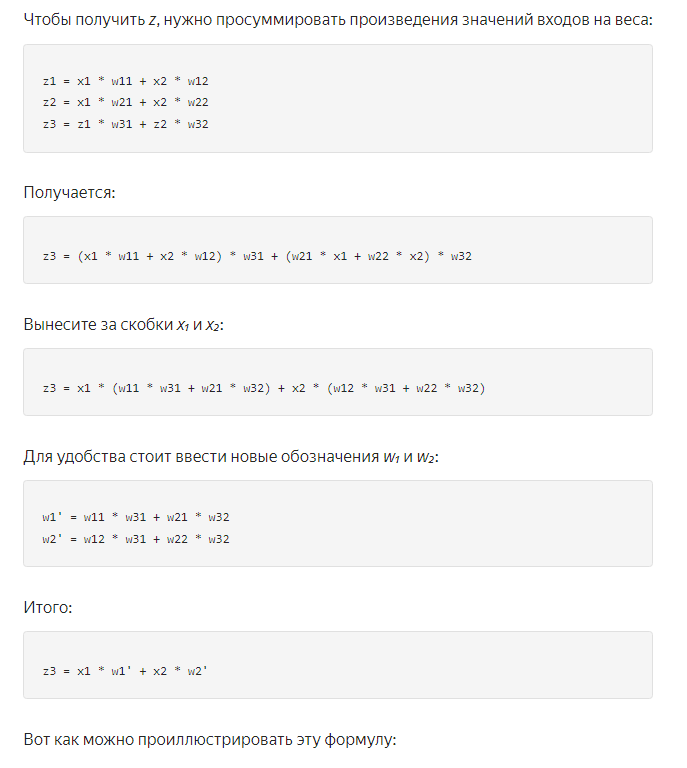


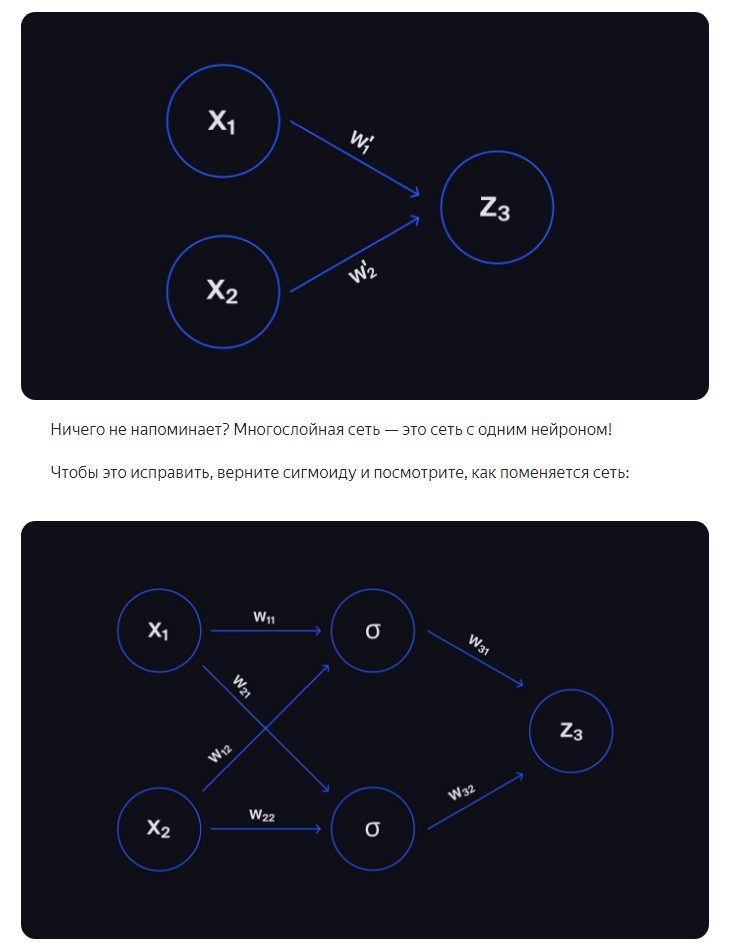
### Полносвязные нейронные сети

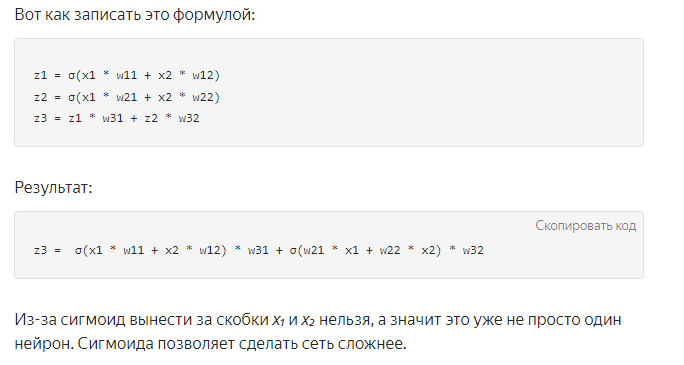
В этом уроке вы познакомитесь с полносвязными нейронными сетями (англ. fully connected neural networks), где в каждом слое любой нейрон связан с каждым нейроном предыдущего слоя.

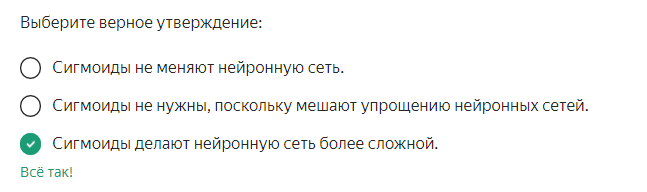
Вот пример такой сети. Скрытыми слоями (англ. hidden layers) называются все слои, кроме входного и выходного.











# Полносвязные нейронные сети в Keras

Полносвязные слои в Keras создаются вызовом метода Dense(). Что нужно сделать, чтобы построить многослойную полносвязную сеть? Несколько раз добавить полносвязный слой! Чем больше слоёв, тем сложнее модель.

Добавьте в нейронную сеть ещё один слой. Пусть у первого скрытого слоя будет 10 нейронов units с сигмоидной активацией. Во втором выходном слое будет один нейрон с сигмоидой: её рассмотрим как вероятность класса «1».

Обучите нейронную сеть на 10 эпохах, напечатав на экране прогресс обучения.

Подсказка

Слои добавляются методом model.add(). Аргумент input\_dim нужно указать только для первого слоя

import pandas as pd

from tensorflow import keras

from sklearn.metrics import accuracy\_score

df = pd.read\_csv('/datasets/train\_data\_n.csv')

df['target'] = (df['target'] > df['target'].median()).astype(int)

features\_train = df.drop('target', axis=1)

target\_train = df['target']

df\_val = pd.read\_csv('/datasets/test\_data\_n.csv')

df\_val['target'] = (df\_val['target'] > df['target'].median()).astype(int)

features\_valid = df\_val.drop('target', axis=1)

target\_valid = df\_val['target']

#model = # < напишите код здесь >

# < напишите код здесь >

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=10, input\_dim=features\_train.shape[1],activation='sigmoid'))

model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1],activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='sgd',metrics=['acc'])

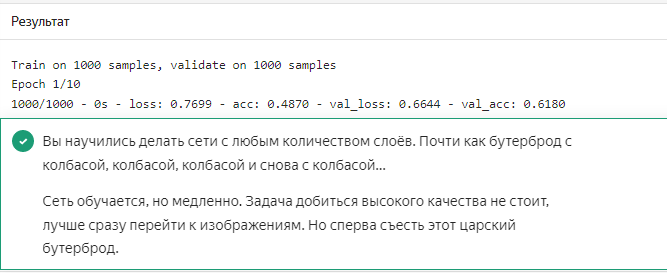
model.fit(features\_train, target\_train, epochs=10, verbose=2,

validation\_data=(features\_valid, target\_valid))

model.compile(loss='binary\_crossentropy', optimizer='sgd', metrics=['acc'])

model.fit(features\_train, target\_train, epochs=10, verbose=2,

validation\_data=(features\_valid, target\_valid))



3.

Постройте и обучите нейронную сеть. Начните с простого — создайте модель логистической регрессии с десятью классами в Keras.

Вам понадобится:

* Функция активации 'softmax';
* Функция потерь 'sparse\_categorical\_crossentropy' (англ. «разрежённая категориальная кросс-энтропия»).

Слово sparse говорит о способе кодирования ответов. В задаче требуется просто номер класса, поэтому выбор пал на эту функцию потерь.

Когда ответы кодируются One-Hot-Encoding и классу 9 соответствует целый вектор [0, 0, 0, 0, 0, 0, 0, 0, 1], применяют categorical\_crossentropy().

Обучите сеть на одной эпохе. Напечатайте прогресс обучения и значения точности на обучающей и тестовой выборках.

from tensorflow import keras

import matplotlib.pyplot as plt

import numpy as np

features\_train = np.load('/datasets/fashion\_mnist/train\_features.npy')

target\_train = np.load('/datasets/fashion\_mnist/train\_target.npy')

features\_test = np.load('/datasets/fashion\_mnist/test\_features.npy')

target\_test = np.load('/datasets/fashion\_mnist/test\_target.npy')

features\_train = features\_train.reshape(features\_train.shape[0], 28 \* 28)

features\_test = features\_test.reshape(features\_test.shape[0], 28 \* 28)

#model = # < напишите код здесь >

model = keras.models.Sequential()

model.add(keras.layers.Dense(units=10, input\_dim=features\_train.shape[1],activation='softmax'))

#model.add(keras.layers.Dense(units=1, input\_dim=features\_train.shape[1],activation='sigmoid'))

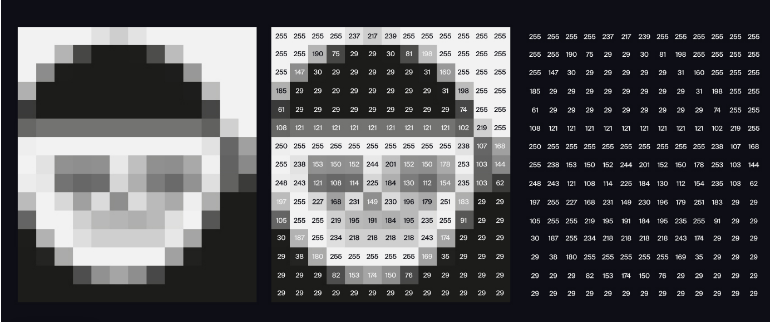
model.compile(loss='sparse\_categorical\_crossentropy', optimizer='sgd',metrics=['acc'])

model.fit(features\_train, target\_train, epochs=1, verbose=2,validation\_data=(features\_test, target\_test))

### Работа с изображениями в Python

Вы уже умеете открывать картинки в Python с помощью библиотеки OpenCV. В этом уроке вы научитесь работать с этими изображениями, переводя их в NumPy-массив.

Вы уже знаете, что изображения — это набор чисел. Если изображение чёрно-белое, то в каждом пикселе хранится число от 0 (чёрный) до 255 (белый).



Для загрузки картинки используем уже знакомую вам библиотеку OpenCV. Затем будем работать с ним, как c NumPy-массивом:

Скопировать кодPYTHON

import numpy as np

from PIL import Image

image = Image.open('/datasets/ds\_cv\_images/face.png')

array = np.array(image)

print(array)



#### Задачи

2.

Изучите функцию imshow() и добавьте аргумент, который сделает цветовую гамму чёрно-белой. Затем добавьте к изображению шкалу цвета вызовом функции colorbar() (англ. «цветовая шкала»):

Скопировать кодPYTHON

plt.colorbar()

Подсказка

За цветовую гамму отвечает аргумент cmap (от англ. *color map*).

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

image = Image.open('/datasets/ds\_cv\_images/face.png')

array = np.array(image)

#plt.imshow(array, cmap='gist\_gray\_r')

plt.imshow(array, cmap='gray')

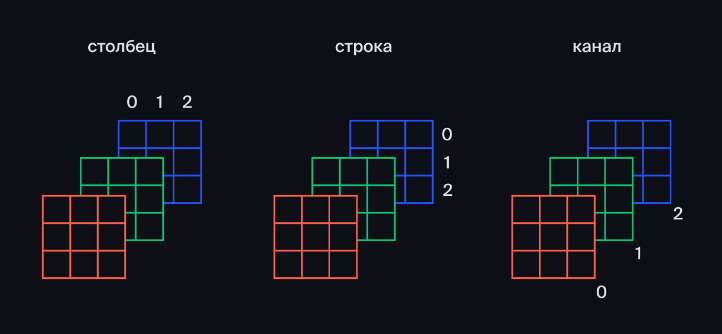
plt.colorbar()

# < напишите код здесь >



### Цветные изображения

С чёрно-белым изображениями всё понятно, а как работать с цветными? Вы уже знаете, что цветные RGB-изображения состоят из трёх каналов: красного, зелёного и синего. По сути такие изображения — это трёхмерные массивы, в ячейках которых могут быть целые числа от 0 до 255.



Трёхмерные массивы в NumPy устроены так же, как и двумерные.

Сравните, как они создаются:

Скопировать кодPYTHON

np.array([[0, 255],

[255, 0]])

Скопировать кодPYTHON

np.array([[[0, 255, 0], [128, 0, 255]],

[[12, 89, 0], [5, 89, 245]]])

В трёхмерном массиве, полученном из изображения, всё так же: первая координата — это номер строки, вторая — номер столбца. Ещё добавляется третья координата — номер канала.

То есть трёхмерный массив — это всё тот же двумерный массив, аналогичный чёрно-белому изображению. Только в каждом пикселе этого массива хранятся три числа: яркость красного, зелёного и синего каналов.

Перейдём к задачам. Фото гангста-кота взяли с сайта: [www.petful.com](http://www.petful.com/),

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

image = Image.open('/datasets/ds\_cv\_images/cat.jpg')

array = np.array(image)

plt.imshow(array)



#### Задачи

1.

Убедитесь, что каналы хранятся в третьей координате. Для этого выведите размер массива, полученного из картинки.

from PIL import Image

import numpy as np

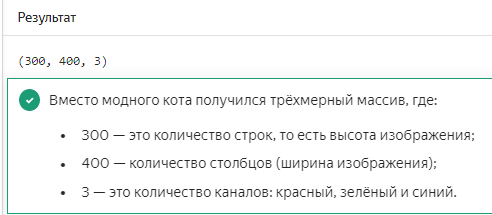
import matplotlib.pyplot as plt

image = Image.open('/datasets/ds\_cv\_images/cat.jpg')

array = np.array(image)

print(array.shape)

# < напишите код здесь >



2.

Изобразите на экране канал только с красным цветом.

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

image = Image.open('/datasets/ds\_cv\_images/cat.jpg')

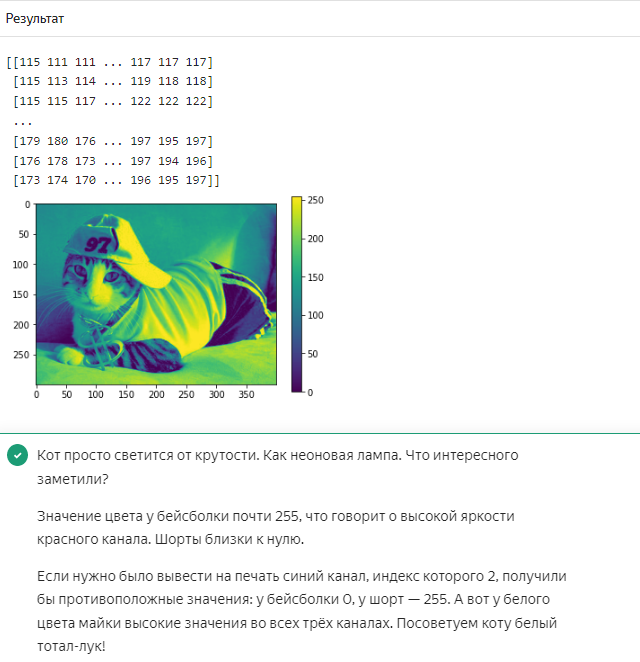
array = np.array(image)

red\_channel = array[:,:,0]

plt.imshow(red\_channel)

plt.colorbar()

print(array[:,:,0])

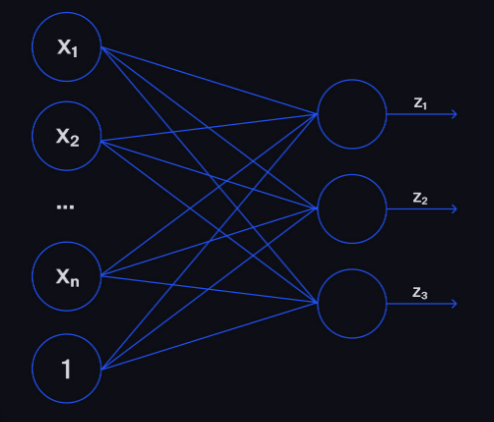


### Многоклассовая классификация

На этом уроке вы разберёте, что такое многоклассовая классификация (англ. multi-class classification) и как она устроена.

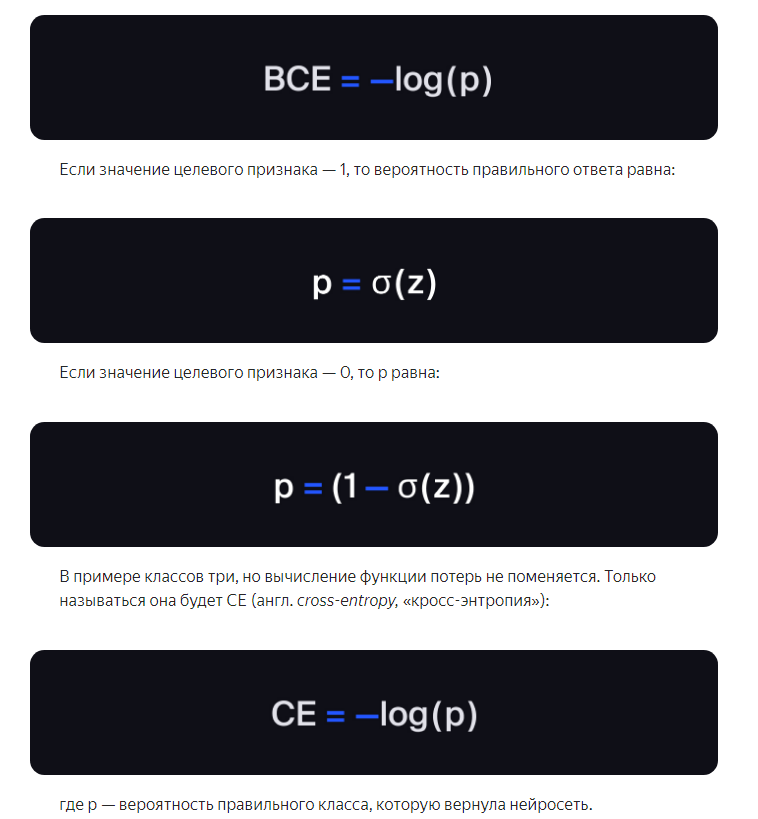
В такой классификации объекты принадлежат не одному из двух классов, а одному из нескольких.

Допустим, вы работаете с тремя классами объектов. Так будет выглядеть логистическая регрессия в виде нейронной сети:

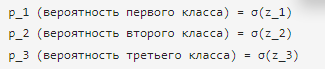


Получилась полносвязная сеть, в выходном слое которой не один, а три нейрона. Каждый нейрон отвечает за свой класс. Если на выходе *z1* будет очень большое положительное значение, то нейронная сеть посчитает, что у объекта класс «1».

Как вычислить функцию потерь? Повторите бинарную кросс-энтропию:



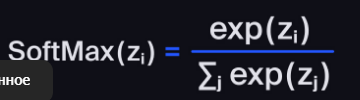
Откуда взять вероятность? Раньше её получали сигмоиды. Что если после каждого нейрона в выходном слое поставить сигмоиду?



Все вероятности варьируются от 0 до 1, но сумма этих вероятностей не обязательно будет равна единице. Если предположить, что объект относится только к одному классу, должно получиться такое равенство:

p\_1 + p\_2 + p3 = 1

Функция активации, которая здесь подойдёт, называется SoftMax (англ. «мягкий максимум»). Она принимает несколько выходов сети и возвращает вероятности, сумма которых равна единице.

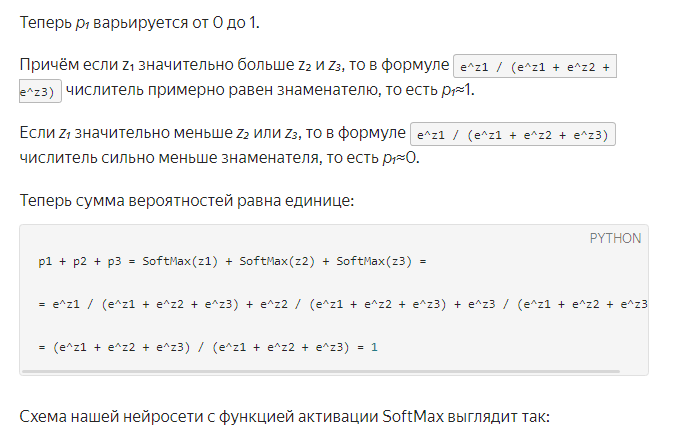


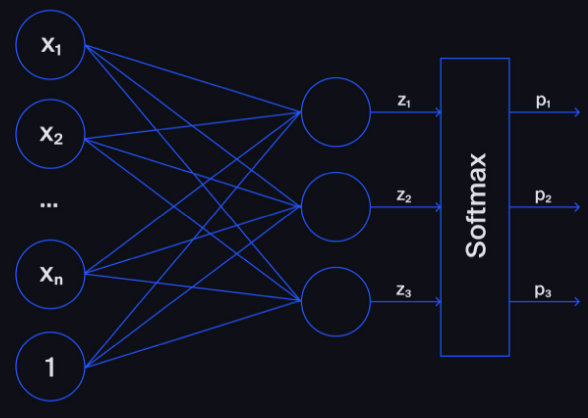
Вероятности будут вычисляться так:

p1 = SoftMax(z1) = e^z1 / (e^z1 + e^z2 + e^z3)

p2 = SoftMax(z2) = e^z2 / (e^z1 + e^z2 + e^z3)

p3 = SoftMax(z3) = e^z3 / (e^z1 + e^z2 + e^z3)



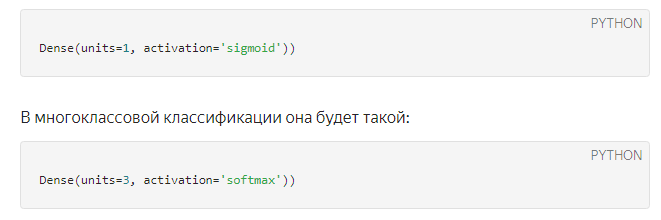


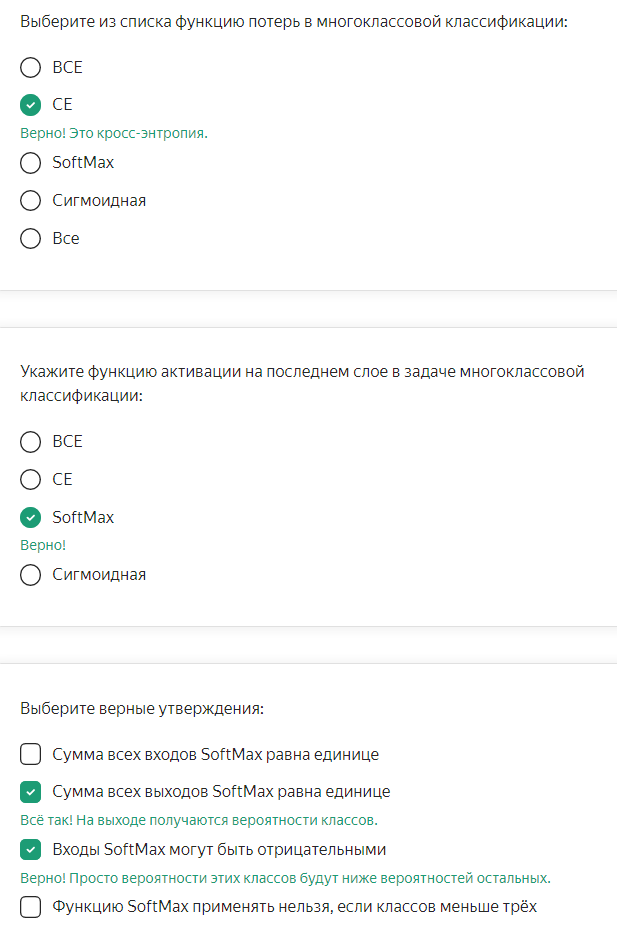
Почему SoftMax изображен как блок, зависящий от всех выходов из сети? Чтобы получить каждую вероятность, действительно нужны все выходы.

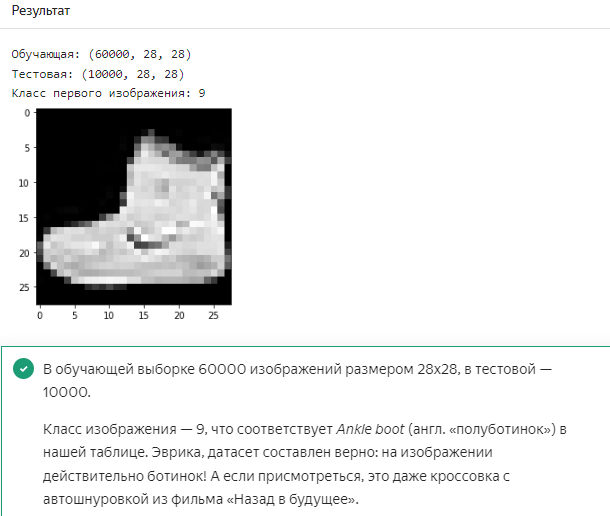
Если классов будет больше трёх, нейронов в выходном слое будет столько же, сколько классов, а все их выходы передадутся в SoftMax.

Вероятности из SoftMax на этапе обучения перейдут в кросс-энтропию, которая и посчитает ошибку. Функция потерь будет минимизирована методом градиентного спуска. Ему достаточно, чтобы у функции была производная по всем параметрам: весам и смещению нейронной сети.

Посмотрите, как изменится код. Если в бинарной классификации инициализация последнего слоя выглядела так:







2.

В полносвязных сетях подаваемые на вход объекты должны быть строками таблицы, а весь датасет — двумерной таблицей. Чтобы не было ошибки, преобразуйте датасет из трёхмерного массива в двумерную таблицу. Для этого вам понадобится метод np.array.reshape() (от англ. «изменить размер»).

Преобразуйте features\_train так, чтобы в первом значении features\_train.shape было количество объектов, а во втором — количество пикселей в изображении. Таким же способом измените features\_test. Напечатайте на экране новые размеры массивов (уже в прекоде).

import matplotlib.pyplot as plt

import numpy as np

features\_train = np.load('/datasets/fashion\_mnist/train\_features.npy')

target\_train = np.load('/datasets/fashion\_mnist/train\_target.npy')

features\_test = np.load('/datasets/fashion\_mnist/test\_features.npy')

target\_test = np.load('/datasets/fashion\_mnist/test\_target.npy')

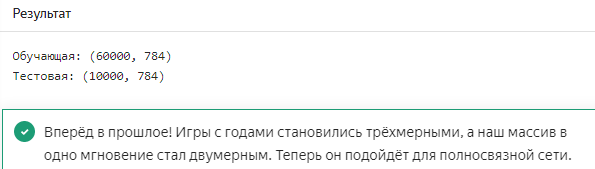
features\_train = features\_train.reshape(features\_train.shape[0], 28 \* 28)

features\_test = features\_test.reshape(features\_test.shape[0], 28 \* 28)

# < напишите код здесь >

print("Обучающая:", features\_train.shape)

print("Тестовая:", features\_test.shape)



### Классификация изображений

Попробуйте решить задачу многоклассовой классификации на изображениях.

В вашем распоряжении [такой датасет](https://github.com/zalandoresearch/fashion-mnist). Для агрегатора магазинов одежды нужно сделать классификатор на 10 классов. Каждый класс будет отвечать за определённую категорию товаров и отражён на сайте агрегатора:

| **Label** | **Description** |
| --- | --- |
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

Вы будете работать с изображениями, потому что описания товаров могут содержать некорректную информацию, их может быть слишком много и причём на разных языках.

На входе модель будет получать чёрно-белую фотографию товара.

#### Задачи

1.

Посмотрите на данные. Мы уже загрузили выборки, разбили их на признаки и целевой признак.

Выведите на экран размеры признаков в обеих выборках. Затем напечатайте значение целевого признака первого изображения из обучающей выборки. Само изображение выведите в чёрно-белом цвете (уже в прекоде).

Подсказка

Допишите код вывода первого изображения:

Скопировать кодPYTHON

print("Класс первого изображения:", *# < напишите код здесь >)*

plt.imshow(features\_train[0], cmap='gray')

import matplotlib.pyplot as plt

import numpy as np

features\_train = np.load('/datasets/fashion\_mnist/train\_features.npy')

target\_train = np.load('/datasets/fashion\_mnist/train\_target.npy')

features\_test = np.load('/datasets/fashion\_mnist/test\_features.npy')

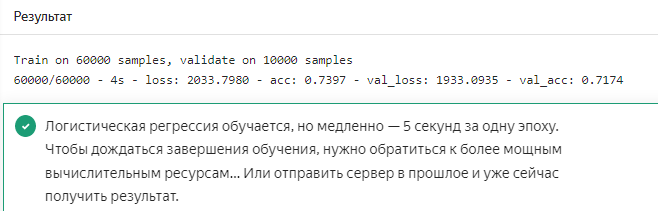
target\_test = np.load('/datasets/fashion\_mnist/test\_target.npy')

print("Обучающая:", features\_train.shape)

print("Тестовая:", features\_test.shape)

print("Класс первого изображения:", target\_train[0])

plt.imshow(features\_train[0], cmap='gray')



### Обучение многослойной сети [GPU]

В этом уроке вы разберётесь, как обучать модели на сервере.

Прежде чем запустить модель на сервере, нужно дождаться своей очереди к нему. Время ожидания зависит от загруженности сервера и может составить 2,5 часа. Это нормально при работе с нейросетями, когда требуются большие вычислительные мощности. Даже в крупных компаниях иногда приходится «подкупать» коллег печеньками, чтобы они уступили очередь.

Чтобы решение задач на GPU не заняло много времени, учтите эти рекомендации:

* Не откладывайте задачи на потом: в конце спринта очередь может быть ещё больше. Всего задач с GPU будет пять.
* Настройте обучение так, чтобы на экран выводилась информация о прогрессе обучения. Это позволит быстрее найти и исправить ошибку.
* Перед отправкой кода на сервер проверьте его на своём компьютере. Не торопитесь: убедитесь, что до обучения модели код действительно работает. Перед тем как поставить задачу в очередь, мы тоже проверим ваш код.

Для предварительной проверки кода создайте в нём три функции:

1. Загрузка обучающей выборки

def load\_train(path):

Загрузите в эту функцию обучающую выборку и командой return верните пару с переменными «признаки и целевой признак». Путь к файлам находится в параметре path.

def load\_train(path):

features\_train = np.load(path + 'train\_features.npy')

target\_train = np.load(path + 'train\_target.npy')

features\_train = features\_train.reshape(features\_train.shape[0], 28 \* 28) / 255.

return features\_train, target\_train

Загружать в эту функцию тестовую выборку не нужно: сервер загрузит её сам.

Яркость изображений из обучающей выборки приведите к диапазону [0, 1]. В этом диапазоне всегда находится яркость изображений из тестовой.

1. Создание модели

def create\_model(input\_shape):

...

Создайте модель для нейронной сети и верните её командой return. Чтобы указать размер данных в первом слое, примените параметр input\_shape.

Вот пример с логистической регрессией:

def create\_model(input\_shape):

model = Sequential()

model.add(Dense(10, input\_shape=input\_shape, activation='softmax'))

model.compile(optimizer='sgd', loss='sparse\_categorical\_crossentropy',

metrics=['acc'])

return model

1. Запуск модели

def train\_model(model, train\_data, test\_data, batch\_size, epochs,

steps\_per\_epoch, validation\_steps):

...

Обучите и верните её командой return.

Функция получает на вход:

* модель,
* обучающую и тестовую выборку,
* аргументы для метода fit().

Аргументы нужны, чтобы перед отправкой на сервер код можно было проверить на небольшом числе объектов. Нужные значения для них запишите как значения по умолчанию.

def train\_model(model, train\_data, test\_data, batch\_size=32, epochs=5,

# эти аргументы разберём позже, поэтому пока не указываем

steps\_per\_epoch=None, validation\_steps=None):

Рассмотрим пример с логистической регрессией:

def train\_model(model, train\_data, test\_data, batch\_size=32, epochs=5,

steps\_per\_epoch=None, validation\_steps=None):

features\_train, target\_train = train\_data

features\_test, target\_test = test\_data

model.fit(features\_train, target\_train,

validation\_data=(features\_test, target\_test),

batch\_size=batch\_size, epochs=epochs,

steps\_per\_epoch=steps\_per\_epoch,

validation\_steps=validation\_steps,

verbose=2, shuffle=True)

return model

Вашей валидационной выборкой в методе fit() будет тестовая.

Посмотрите на весь код для логистической регрессии:

def load\_train(path):

features\_train = np.load(path + 'train\_features.npy')

target\_train = np.load(path + 'train\_target.npy')

features\_train = features\_train.reshape(features\_train.shape[0], 28 \* 28) / 255.

return features\_train, target\_train

def create\_model(input\_shape):

model = Sequential()

model.add(Dense(10, input\_shape=input\_shape, activation='softmax'))

model.compile(optimizer='sgd', loss='sparse\_categorical\_crossentropy',

metrics=['acc'])

return model

def train\_model(model, train\_data, test\_data, batch\_size=32, epochs=5,

steps\_per\_epoch=None, validation\_steps=None):

features\_train, target\_train = train\_data

features\_test, target\_test = test\_data

model.fit(features\_train, target\_train,

validation\_data=(features\_test, target\_test),

batch\_size=batch\_size, epochs=epochs,

steps\_per\_epoch=steps\_per\_epoch,

validation\_steps=validation\_steps,

verbose=2, shuffle=True)

return model

Чтобы выполнить задание, перейдите на курс «Компьютерное зрение — практика обучения нейронных сетей», выберите тему «Полносвязные сети», а в теме урок «Обучение многослойной сети [GPU]».