Д.К.МУХАМЕДИЕВА, Н.А.НИЁЗМАТОВА,
М.Ю.ДОШАНОВА

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
ПО ДИСЦИПЛИНЕ

ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

MUHAMMAD AL-XORAZMIY NOMIDAGI МИНИСТЕРСТВО ПО
РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ
РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

Д.К.МУХАМЕДИЕВА, Н.А.НИЁЗМАТОВА, М.Ю.ДОШАНОВА

Учебно-методическое пособие по дисциплине

«ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ»

ТАШКЕНТ 2022
ИЗДАТЕЛЬСТВО «FAN ZIYOSI»

УДК 344.428.14
КВК 26.325
М-18

Д.К.Мухамедиева, Н.А.Ниёзматова, М.Ю.Дошанова.
Учебно-методическое пособие по дисциплине «Предварительная обработка данных» – Ташкент.: Изд. «Fan ziyosi», 2022. - 88 стр. с ил.

Учебно-методическое пособие предназначено для преподавателей и студентов высших учебных заведений по направлению «5330600–Программный инжиниринг».

Рецензент:
У.Т.Мухамедханов – Профессор кафедры «Автоматизация производственных процессов» ТГТУ, д.т.н.

Напечатано по решению учебно–методического совета ТУИТ
(протокол № 9(155) от «26» апрел 2022)

ISBN 978-9943-876-64-4

Введение

Цель курса направлен иориентирован на получение знаний по технологиям разработки методов, предварительной обработки данных, основанных на исследованиях и разработок отечественных и зарубежных исследователей, а также охватывает алгоритмы мягких вычислений, которые являются основным механизмом систем предварительной обработки данных

Следующие требования предъявляются к знаниям, навыкам и умениям студентов по предмету. Студент дожен:

Знать классы задач, решаемых с помощью базы данных, программирование; основные виды интеллектуального анализа данных; способы представления знаний в интеллектуальных системах; алгоритмы логического вывода на знаниях; принцип действия интеллектуальных систем на нейронных сетях.

Знать модели представления знаний; понятие и структуру задач классификации; понятие и принципы работы задач кластеризации; основы функционирования информационно-поисковых систем; основные сведения о языках программирования искусственного интеллекта.

Уметь: описывать и создавать базу данных по требуемой предметной области; решать поставленные задачи в условиях нечеткой исходной информации; описывать и строить базы знаний; организовать поисковую интеллектуальную информационную систему.

Владеть: терминологией в предметной области анализа данных; навыками решения логических задач с использованием языка Python; навыками решения задач с нечеткими числовыми данными; навыками использования средств интеллектуализации в решении задач автоматизированного проектирования и создания технологий для его поддержки.

Цель лабораторной работы: познакомиться со средой разработки Python. Изучить основные типы данных, команды ввода и вывода данных.

Краткая теория

Python– это объектно-ориентированный, интерпретируемый, переносимый язык сверхвысокого уровня. Программирование на Python позволяет получать быстро и качественно необходимые программные модули.

В комплекте вместе с интерпретатором Python идет IDLE (интегрированная среда разработки). По своей сути она подобна интерпретатору, запущенному в интерактивном режиме с расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка и т.п.).

Для запуска IDLE в Windows необходимо перейти в папку Python в меню "Пуск" и найти там ярлык с именем "IDLE (Python 3.X XX-bit)".

Для запуска редактора программы (кода) следует выполнить команду File->New File или сочетание клавиш Ctrl+N.

Любая Python-программа состоит из последовательности допустимых символов, записанных в определенном порядке и по определенным правилам.

Программа включает в себя:

- комментарии;
- команды;
- знаки пунктуации;
- идентификаторы;
- ключевые слова.

Комментарии в Python обозначаются предваряющим их символом # и продолжаются до конца строки(т.е. в Python все комментарии являются однострочными), при этом не допускается использование перед символом # кавычек:

Знаки пунктуации

В алфавит Python входит достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки "+" или " *" могут использоваться для сложения и умножения, а знак запятой "," - для разделения параметров функций.

Идентификаторы

Идентификаторы в Python это имена используемые для обозначения переменной, функции, класса, модуля или другого объекта.

Ключевые слова

Некоторые слова имеют в Python специальное назначение и представляют собой управляющие конструкции языка.

Ключевые слова в Python:

['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

Типы данных

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)
3. Числа (Numeric Type)
    1. int – целое число
    2. float – число с плавающей точкой
    3. complex – комплексное число
4. Списки (Sequence Type)
    1. list – список
    2. tuple – кортеж
    3. range – диапазон
5. Строки (Text Sequence *Type*)
    1. str

Ввод и вывод данных

Ввод данных осуществляется при помощи команды **input** (список ввода):

```
a = input()
print(a)
```

В скобках функции можно указать сообщение - комментарий к вводимым данным:

```
a = input ("Введите количество: ")
```

Команда input() по умолчанию воспринимает входные данные как строку символов. Поэтому, чтобы ввести целочисленное значение, следует указать тип данных int():

```
a = int (input())
```

Для ввода вещественных чисел применяется команда

```
a=float(input())
```

Вывод данных осуществляется при помощи команды **print**(список вывода):

```
a = 1
b = 2
print(a)
print(a + b)
print('сумма = ', a + b)
```

Существует возможность записи команд в одну строку, разделяя их через;. Однако не следует часто использовать такой способ, это снижает удобочитаемость:

```
a = 1; b = 2; print(a)
print (a + b)
print ('сумма = ', a + b)
```

Для команды **print** может задаваться так называемый сепаратор — разделитель между элементами вывода:

```
x=2
y=5
print ( x, "+", y, "=", x+y, sep = " " )
```

Результат отобразится с пробелами между элементами: 2 + 5 = 7

Простые арифметические операции над числами

x + y	Сложение
x - y	Вычитание
x * y	Умножение
x / y	Деление

Пример программы на Python

Результат выполнения программы с применением простых арифметических операций

Для форматированного вывода используется **format**:

Строковый метод format() возвращает отформатированную версию строки, заменяя идентификаторы в фигурных скобках {}. Идентификаторы могут быть позиционными, числовыми индексами, ключами словарей, именами переменных.

Синтаксис команды **format**:

поле замены := "{" [имя поля] ["!" преобразование] [":" спецификация] "}"

имя поля := arg_name ("." имя атрибута | "[" индекс "]")*

преобразование := "r" (внутреннее представление) | "s" (человеческое представление)

спецификация := см. ниже

Аргументов в format() может быть больше, чем идентификаторов в строке. В таком случае оставшиеся игнорируются.

Идентификаторы могут быть либо индексами аргументов, либо ключами:

В результате выведется число 11, а перед ним два пробела, так как указано использовать для вывода четыре знакоместа.

Или с несколькими аргументами:

В итоге каждое из значений выводится из расчета 4 знакоместа.

Спецификация формата:

спецификация	:= [[fill]align][sign][#][0][width][.][.precision][type]
заполнитель	:= символ кроме '{' или '}'
выравнивание	:= "<" | ">" | "=" | "^"
знак	:= "+" | "-" | " "
ширина	:= integer
точность	:= integer
тип	:= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).

'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре). если она меньше, чем -4 или точности. иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

Для форматирования вещественных чисел с плавающей точкой используется следующая команда:

print('{0:.2f}'.format(вещественное число))

В результате выведется число с двумя знаками после запятой.

Пример

Напишите программу, которая запрашивала бы у пользователя:

Вариант 0

- ФИО ("Ваши фамилия, имя, отчество?")
- возраст ("Сколько Вам лет?")
- место жительства ("Где вы живете?")

После этого выводила бы три строки:

"Ваше имя"

"Ваш возраст"

"Вы живете в"

Задания для самостоятельной работы (по вариантам)

Напишите программу, которая запрашивала бы у пользователя:

Вариант 1

Имя, Фамилия, Возраст, Место жительства

- фамилия, имя ("Ваши фамилия, имя?")
- возраст ("Сколько Вам лет?")
- место жительства ("Где вы живете?")

После этого выводила бы три строки:

"Ваши фамилия, имя"
"Ваш возраст"
"Вы живете в"

Вариант 2
Имя, , Дата рождения, Образование
- имя ("Ваше, имя?")
- дата рождения ("Ваша дата рождения?")
- образование ("Где Вы учитесь?")
После этого выводила бы три строки:
"Ваше имя"
"Дата рождения"
"Вы учитесь в "

Вариант 3
Фамилия, Место жительства
- Фамилия("Ваша фамилия?")
- место жительства ("Где Вы живете?")
После этого выводила бы две строки:
"Ваша фамилия"
"Вы живете в"

Вариант 4
Фамилия, Место рождения, любимая музыка
- Фамилия, ("Ваша фамилия?")
- место рождения ("Где Вы родились?")
- музыка("Какая музыка нравится? ")
После этого выводила бы три строки:
"Ваши имя, фамилия"
"Вы родились в"
"Ваша любимая музыка "
Вариант 5
Имя, Фамилия, ФИО мамы, ФИО отца
- ФИО (например, "Ваши фамилия, имя, отчество?")
- возраст ("Сколько Вам лет?")
- место жительства ("Где Вы живете?")
После этого выводила бы три строки:
"Ваши имя, фамилия, отчество"
"Ваш возраст"

"Вы живете в"

Вариант 6

Имя, Любимый предмет в школе, Номер класса

- имя ("Ваше имя?")
- любимый предмет ("Какой Ваш любимый предмет в школе?")
- номер класса ("В каком классе Вы учитесь?")

После этого выводила бы три строки:

"Ваше имя"

"Ваш любимый предмет в школе"

"Вы учитесь в классе номер"

Вариант 8

Имя, Фамилия, Отчество, Хобби

- ФИО (например, "Ваши фамилия, имя, отчество?")
- хобби ("Чем Вы увлекаетесь?")

После этого выводила бы две строки:

"Ваши имя, фамилия, отчество"

"Ваше хобби"

Вариант 9

Имя, Фамилия, любимый спорт

- Фамилия, имя ("Ваши фамилия, имя?")
- образование ("В какой школе Вы учитесь?")
- ФИО Вашего руководителя по информатики ("ФИО Вашего руководителя по информатики?")

После этого выводила бы три строки:

"Ваши имя, фамилия"

"Вы учитесь в школе номер: "

"ФИО Вашего руководителя по информатике "

Вариант 10

Имя, Фамилия, Любимый предмет в школе (в институте). ФИО классного руководителя (куратора)

- Фамилия, имя ("Ваши фамилия, имя?")
- любимый предмет в школе ("Какой Ваш любимый предмет в школе?")
- ФИО классного руководителя ("ФИО Вашего классного руководителя?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"Ваш любимый предмет в школе "
"ФИО Вашего классного руководителя"

Вариант 11

Имя, Фамилия, Возраст, Дата рождения
- Фамилия, имя ("Ваши фамилия, имя?")
- возраст ("Сколько Вам лет?")
- дата рождения ("Когда Вы родились?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"Ваш возраст"
"Дата Вашего рождения"

Вариант 12

Имя, Фамилия, Место жительства, Месторождения
- Фамилия, имя ("Ваши фамилия, имя?")
- место рождения ("Где Вы родились?")
- место жительства ("Где Вы живете?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"Вы родились в"
"Вы живете в"

Вариант 13

Имя, Фамилия, Возраст, Номер телефона
- Фамилия, имя ("Ваши фамилия, имя?")
- возраст ("Сколько тебе лет?")
- номер телефона ("Номер Вашего телефона?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"Ваш возраст"
"Ваш номер телефона"

Вариант 14

Имя, Фамилия, Страна, Край , Город
- Фамилия, имя ("Ваши фамилия, имя?")
- страна ("В какой стране Вы живете?")
- город ("В каком городе Вы живете?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"Вы живете в стране"
"Вы живете в крае"
"Вы живете в городе"

Вариант 15

Имя, Фамилия, ФИО Вашего классного руководителя
- Фамилия, имя ("Ваши фамилия, имя?")
- ФИО Вашего классного руководителя ("ФИО Вашего классного руководителя?")

После этого выводила бы три строки:
"Ваши имя, фамилия"
"ФИО Вашего руководителя по информатике"
"ФИО Вашего классного руководителя"

Список литературы

1. https://www.opennet.ru/docs/RUS/python/python_b.html
2. https://metanit.com/python/tutorial/1.1.php
3. https://pythonworld.ru/osnovy

Лабораторная работа №2. Формирование таблицы об распределении данных

Популярная библиотека Pandas предоставляет несколько различных вариантов визуализации с помощью функции **.plot()**.

Чтобы лучше взаимодействовать с материалом статьи, можно использовать Jupyter Notebook -- вы сразу увидите свои графики и сможете поиграть с ними.

Вам также понадобится рабочая среда Python вместе с Pandas. Если она еще не установлена, есть варианты, как это исправить:

- Если у вас есть далекоидущие планы, скачайте дистрибутив Anaconda. Он весит ~500 МБ, но в него включено практически все, что не обходимо для DS.
- Если предпочитаете минимальную установку – ознакомьтесь с Miniconda-ой.
- Если любите pip, то установите библиотеки, с помощью pip install pandas matplotlib и Jupyter с помощью pip install jupyterlab.
- Если же не хотите совершать лишних телодвижений, воспользуйтесь Jupyter онлайн.

Встатьеиспользуютсяданныересурса: Economic Guide To Picking A College Major. Как только настроите среду, загрузите набор данных, передав URL-адрес загрузки в **pandas.read_csv()**:

Вызывая **read_csv()**, вы создадите DataFrame – основную структуру данных в Pandas.

Теперь, когда есть фрейм данных, можно его изучить. Настроим параметр **display.max.columns**, чтобы убедиться, что Pandas не скрывает никаких столбцов. Просмотреть первые несколько строк данных можно с помощью .head():

Вывод должен выглядеть следующим образом:

	Rank	Major_code	Major	Total	Men	Women	Major_category	ShareWomen	Sample_size	Employed	...	Part_time	Full_time_year_round
0	1	2419	PETROLEUM ENGINEERING	2339.0	2057.0	282.0	Engineering	0.120564	36	1976		270	12
1	2	2416	MINING AND MINERAL ENGINEERING	756.0	679.0	77.0	Engineering	0.101852	7	640		170	3
2	3	2415	METALLURGICAL ENGINEERING	856.0	725.0	131.0	Engineering	0.153037	3	648		133	3
3	4	2417	NAVAL ARCHITECTURE AND MARINE ENGINEERING	1258.0	1123.0	135.0	Engineering	0.107313	16	758		150	6
4	5	2405	CHEMICAL ENGINEERING	32260.0	21239.0	11021.0	Engineering	0.341631	289	25694		5180	168

5 rows × 21 columns

По умолчанию функция .head() отображает пять строк, но в качестве аргумента указывается любое их количество. Например, для отображения первых десяти можно использовать df.head(10).

Набор данных содержит несколько столбцов:

- Median – средний заработок работающих полный день круглый год.
- P25th – 25-й процентиль дохода.
- P75th – 75-й процентиль дохода.
- Rank – ранг по среднему заработку.

Начнем с отображающего эти столбцы графика. Настроим Jupyter с помощью волшебной команды %matplotlib:

Эта команда сообщает Jupyter, что далее отображение графиков будет происходить с помощью Matplotlib.

Вы можете изменить бекенд Matplotlib путем передачи аргумента команде %matplotlib. Встроенный (inline) бекенд популярен для Jupyter Notebooks, поскольку он отображает график в самом блокноте, непосредственно под ячейкой, которая его создает:

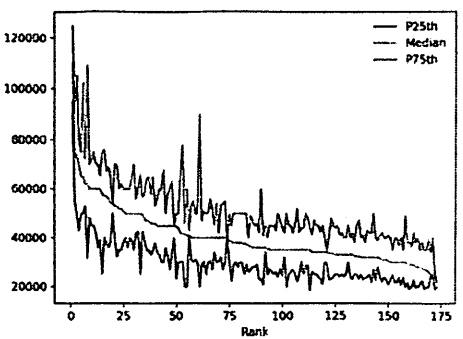
Ознакомиться с другими доступными вариантами бекендов можно на странице IPython.

Теперьсоздадимграфик:

plot() возвращает линейный график, содержащий данные из каждой строки в DataFrame. По оси X выводится рейтинг учреждений, а значения отображаются на оси Y.

Если вы не используете Jupyter Notebook или оболочку IPython, возьмите интерфейс pyplot из matplotlib для отображения графика.

Вот как должно получиться в стандартной оболочке Python (для вызова **plt.show()** импортируйте модуль pyplot из Matplotlib):



Глядя на график, можно сделать следующие наблюдения:

- Средний доход уменьшается с уменьшением рейтинга. Это ожидаемо, потому что он определяется медианным доходом.
- Некоторые специальности имеют большие промежутки между 25-м и 75-м процентилями. Люди с такими степенями могут зарабатывать значительно меньше или значительно больше, чем средний доход.
- Другие специальности имеют очень небольшие промежутки между 25-м и 75-м процентилями. Люди с такими степенями получают зарплату, очень близкую к среднему доходу.

Некоторые специальности имеют широкий диапазон заработка, а другие – довольно узкий. Чтобы обнаружить и визуализировать эти различия, будем использовать другие типы графиков.

.plot() имеет несколько необязательных параметров, определяющих, какой тип графика создается:

- «area» – графики области;
- «bar» – вертикальные графики;
- «barh» – горизонтальные графики;
- «box» – квадратный график;
- «hexbin» – hexbin участки;
- «hist» – гистограммы;
- «kde» – оценка плотности ядра;
- «density» – псевдоним для «kde»;
- «line» – линейные графики;
- «pie» – круговые диаграммы;
- «scatter» – точечные диаграммы.

Значение по умолчанию – «line». Линейные графики обеспечивают хороший обзор ваших данных. Если не задать параметр для функции .plot(), она создаст линейный график с индексом по оси X и всеми числовыми столбцами по оси Y. Хотя это полезное значение по умолчанию для наборов данных с несколькими столбцами, для нашего датасета и его нескольких числовых столбцов оно выглядит небрежно.

В качестве альтернативы передаче строк параметру kind функции .plot(), объекты DataFrame имеют несколько методов, которые можно использовать для создания различных типов графиков:

- .area()
- .bar()
- .barh()
- .box()
- .hexbin()

- .hist()
- .kde()
- .density()
- .line()
- .pie()
- .scatter()

По возможности попробуйте эти методы в действии.

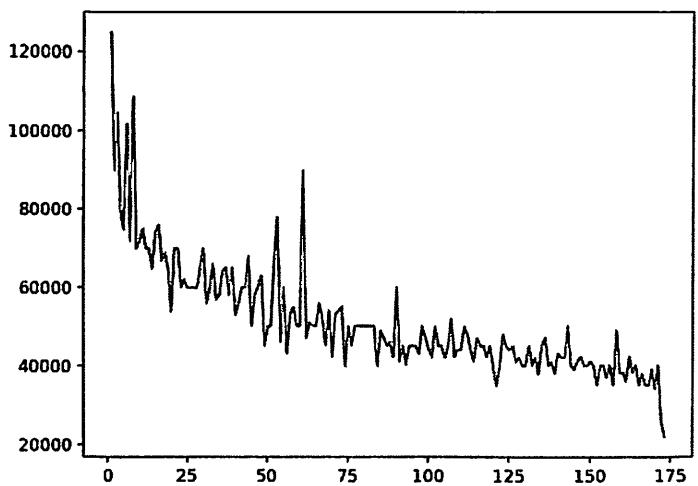
Теперь, когда мы создали график, рассмотрим подробнее работу функции .plot().

Когда вы вызываете функцию .plot() для объекта DataFrame, Matplotlib создает график.

Чтобы убедиться в этом, воспользуемся двумя фрагментами кода. Создайте график с помощью Matplotlib, используя два столбца DataFrame:

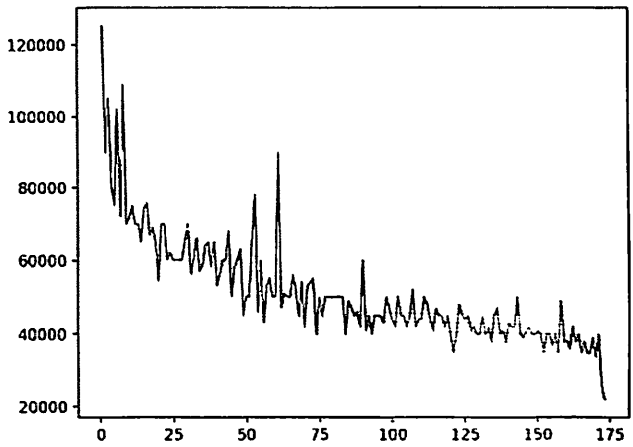
Сначала импортируется модуль **matplotlib.pyplot** и переименовывается в plt. Затем вызывается plot() и передается объект «Rank» в качестве первого аргумента, и «P75th» в качестве второго.

В результате получается линейный график, который строит 75-й процентиль по оси Y и рейтинг по оси X:



Вы можете создать такой же график, используя метод DataFrame:

.plot() – этооболочкадля pyplot.plot(). На выходе получается график, идентичный тому, который мы создали на Matplotlib:



Вы можете использовать как **pyplot.plot()**, так и **df.plot()** для создания одного и того же графика. Однако если у вас уже есть экземпляр DataFrame, то df.plot() предлагает более чистый синтаксис.

Теперь приступим к изучению различных типов графиков и способов их создания.

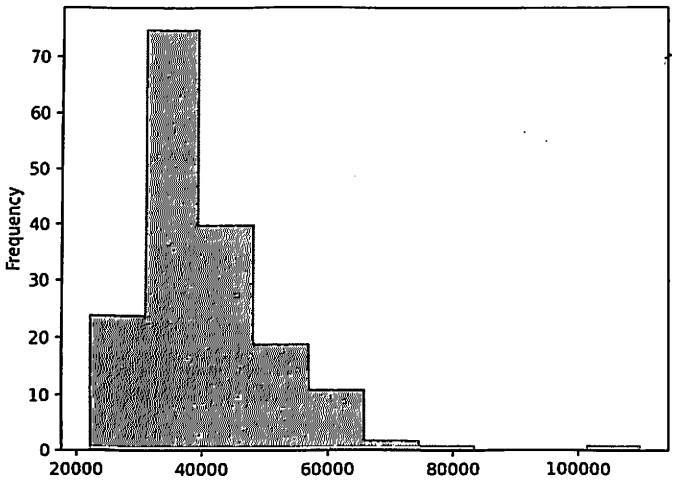
Следующие графики дадут общий обзор конкретного столбца набора данных. Вы рассмотрите распределение свойств на гистограмме и познакомитесь с инструментами для изучения исключений.

DataFrame – не единственный класс в pandas с методом .plot(). Объект Series предоставляет аналогичную функциональность. Вы можете получить каждый столбец DataFrame как объект Series. Вот пример использования столбца «Median»:

Гистограммы – это хороший способ визуализации распределения значений по набору данных. Они группируют значения в ячейки и отображают количество точек данных, значения которых находятся в определенной ячейке.

Создадим гистограмму для столбца «Median»:

Вызывается функция .plot() в median_column и передается строк в «hist» параметру kind. На выходе должно получиться следующее:



Гистограмма показывает данные, сгруппированные в десять ячеек в диапазоне от $20 000 до $120 000 с шагом в $10 000.

В правой части графика виднеется маленькая ячейка – специалисты в этой области получают большую зарплату по сравнению со всеми категориями. Хотя это не основная цель, гистограмма может помочь обнаружить такие исключения. Исследуем эту штуку подробнее:

- какие специальности представляет это исключение?
- где его граница?

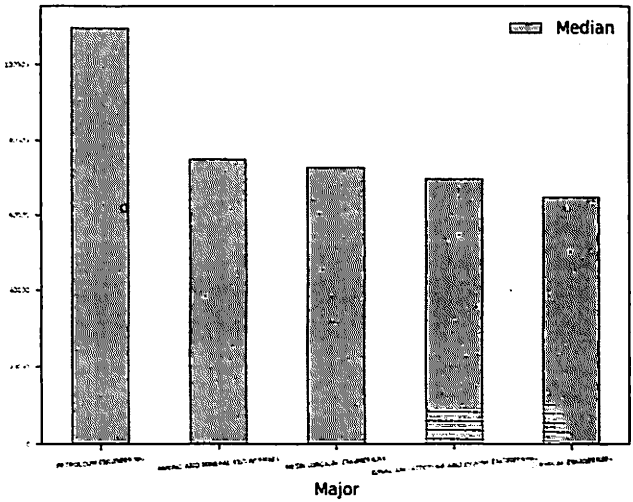
В отличие от первого графика, мы хотим сравнить несколько точек данных и увидеть более подробную информацию о них. Для этого вертикальный график является отличным инструментом. Выберем пять специальностей с самым высоким средним доходом. Необходимо выполнить два шага:

1. Для сортировки по столбцу «Median» используйте функцию .sort_values() и укажите имя столбца, а также направление ascending=False.
2. Чтобы получить первые пять пунктов списка, используйте функцию .head().

СоздадимновыйDataFrametop_5:

Создадим гистограмму, показывающую основные специальности, учитывая ТОП-5 зарплат:

Мы используем параметры rot и fontsize для поворота и изменения размера меток оси X, чтобы они были видны:



Этот график показывает, что средняя зарплата специалистов нефтегазового направления более чем на $20 000 выше остальных. Заработки занявших 2-4 места специальностей относительно близки друг к другу.

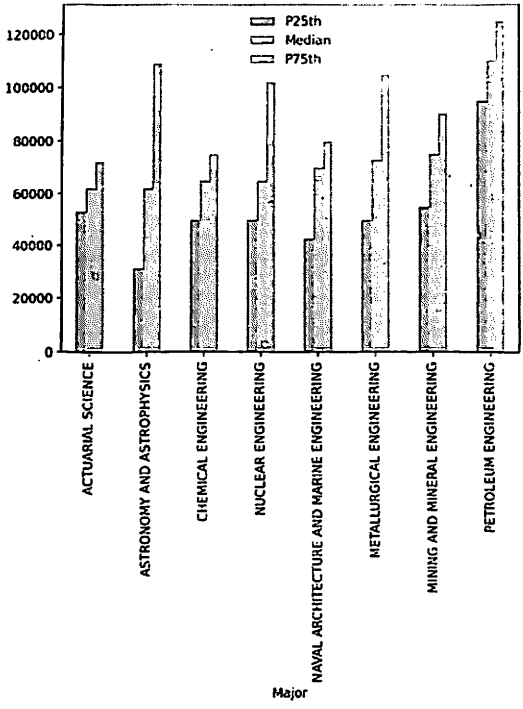
Если у вас есть точки данных с гораздо более высокими/низкими значениями, чем остальные, необходимо этот момент исследовать: можно просмотреть столбцы, содержащие связанные данные.

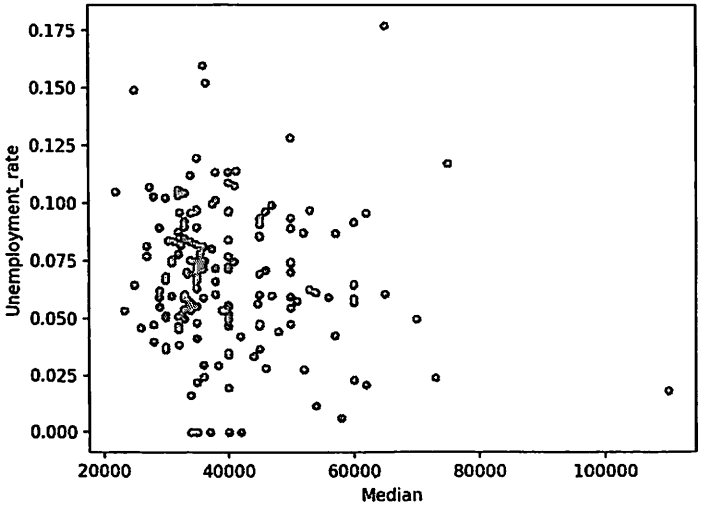
Рассмотрим все специальности, средняя зарплата которых превышает $60 000. Отфильтруем их по маске df[df["Median"] > 60000] и создадим график с тремя столбцами:

25-й и 75-й процентили подтверждают, что нефтяники самые высокооплачиваемые работники.

Почему исключения так важны? Если вы студент – все очевидно, но исключения интересны с точки зрения анализа. Неверные данные могут быть вызваны ошибками или погрешностями.

Часто требуется проверить, связаны ли столбцы набора данных. Например, связан ли высокий оклад с вероятностью не получить работу. В качестве первого шага создайте точечную диаграмму с этими столбцами:


P25th
Median
P75th
120000
100000
80000
60000
40000
20000
0
ACTUARIAL SCIENCE
ASTRONOMY AND ASTROPHYSICS
CHEMICAL ENGINEERING
NUCLEAR ENGINEERING
NAVAL ARCHITECTURE AND MARINE ENGINEERING
METALLURGICAL ENGINEERING
MINING AND MINERAL ENGINEERING
PETROLEUM ENGINEERING
Major


Unemployment_rate
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
20000
40000
60000
80000
100000
Median

Можно заметить, что существенной корреляции между доходами и уровнем безработицы нет.

Хотя точечная диаграмма является отличным инструментом для получения первого впечатления о возможной корреляции, она не является окончательным доказательством связи – для этого подойдет функция .corr().

Однако имейте в виду, что даже если существует корреляция между двумя значениями, не факт, что изменение одного приведет к изменению другого.

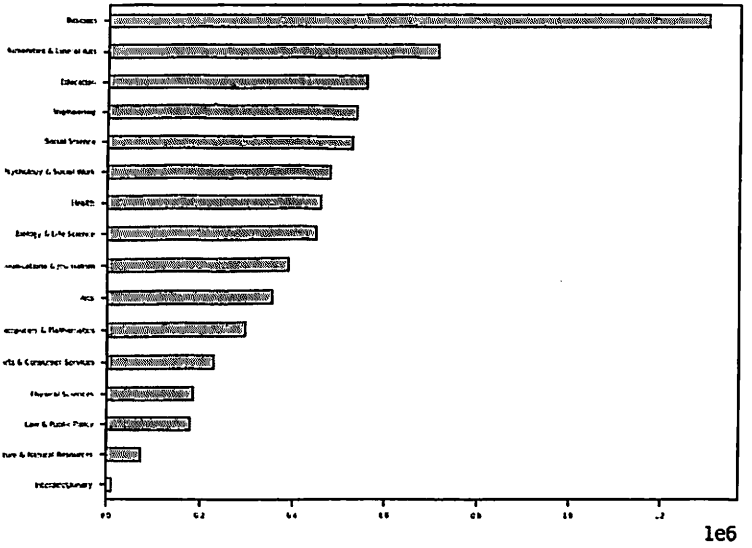
Чтобы обрабатывать большие куски информации, удобно сортировать ее по категориям. Здесь мы познакомимся с инструментами для оценки категорий и проверки ее валидности.

Многие наборы данных уже содержат явную или неявную категоризацию – в нашем примере 173 специальности разделены на 16 категорий.

Основное использование категорий – группирование и агрегирование. Можно использовать функцию .groupby() для определения популярности каждой из категорий в основном датасете:

С помощью функции **.groupby()** создадим объект DataFrameGroupBy. С помощью **.sum()** получим Series.

Нарисуем горизонтальную диаграмму со значениями cat_totals:

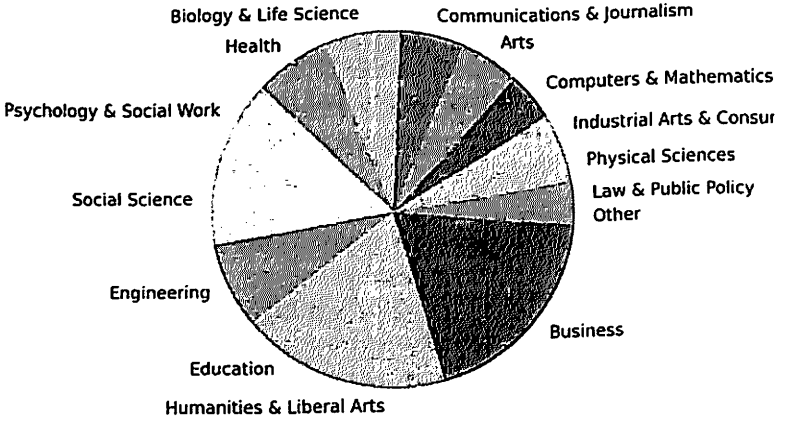


Если необходимо визуализировать соотношения, пригодятся круговые графики. Поскольку cat_totals содержит несколько маленьких категорий,

создание кругового графика с помощью cat_totals.plot(kind="pie") приведет к появлению крошечных фрагментов с перекрывающимися метками.

Чтобы решить проблему, следует объединить более мелкие категории в одну группу. Например, категории с общим числом менее 100 000 в категорию «Другое». Теперь создадим круговую диаграмму:

Обратите внимание, что мы использовали label="". По умолчанию Pandas добавляет метку с именем столбца. Это часто имеет смысл, но в данном случае будет только отвлекать. На выходе получим следующий результат:



При анализе данных очень важна наглядность. Чаще всего недостаточно просто собрать что-то и сгруппировать — данные ещё нужно показать. А ещё лучше отобразить данные так, чтобы одного взгляда было достаточно для принятия решений.

Для визуализации данных в Python чаще всего используют библиотеки Matplotlib, ggplot, Bokeh, pygal, seaborn. А как их использовать, в каком случае и использовать ли вообще, сейчас разберёмся.

Подключим библиотеки Matplotlib и seaborn к проекту:

```
from math import pi
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use('ggplot')
importseabornassns
```

Сначала нам нужны данные для экспериментов: я нашла несколько вакансий на должность веб-разработчика в разных странах и взяла предлагаемую зарплату. Получилисьтакиесловари:

```
salary_2017 = {
```

```
 "USA": "1400",
 "China": "1100",
 "Russia": "1350",
 "Germany": "1300",
 "Hungary": "1200",
 "Ukraine": "1200",
 "France": "1400",
 "Sweden": "1500",
 "UK": "1250",
 "Spain": "1300"
}

salary_2018 = {
 "USA": 1700,
 "China": 1300,
 "Russia": 1450,
 "Germany": 1400,
 "Hungary": 1500,
 "Ukraine": 1300,
 "France": 1500,
 "Sweden": 1600,
 "UK": 1500,
 "Spain": 1400
}
```

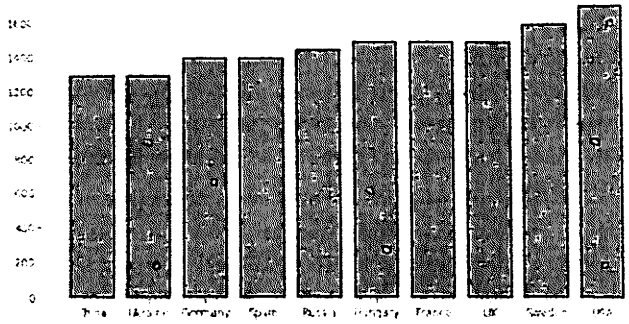
Теперь нужно решить что именно мы хотим узнать:

- Какие у нас есть страны и зарплаты? (столбчатая диаграмма)
- Какая самая распространенная зарплата для веб-разработчика? (гистограмма или тепловая карта)
- Насколько разные зарплаты предлагают? (диаграмма размаха)
- Как стать веб-разработчиком? (радиальная диаграмма)

Столбчатая диаграмма — это как таблица, но для презентаций. У нас есть данные по странам за разные годы, выберем нужный словарь и выведем:

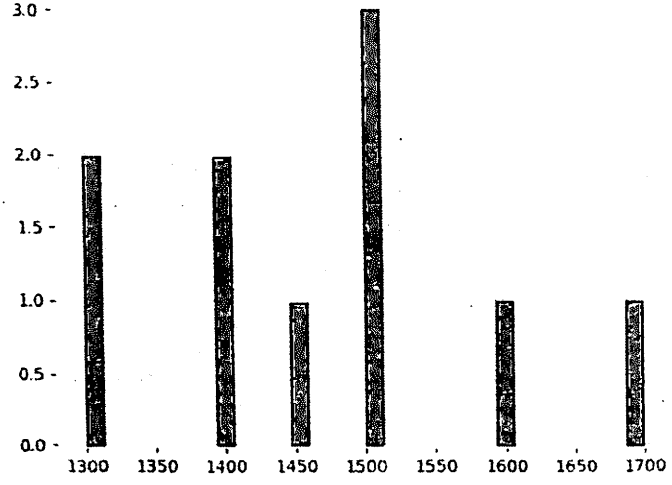
```
d = OrderedDict(sorted(salary_2018.items(), key=lambda x: x[1]))
values = list(d.values())
keys = list(d.keys())
plt.bar(range(len(d)),values,tick_label=keys)
```

plt.show()



Гистограмма — один из вариантов столбчатой диаграммы для оценки распределения значений. Можно оценить частоту появления одного значения или наложить несколько для сравнения. Посмотримраспределениезарплат:

```
data = list(salary_2018.values())
plt.hist(data, bins=30)
plt.show(
```



Тепловая карта разворачивает гистограмму в другую плоскость. Здесь мы цветом показываем частоту разных значений. Посмотрим количество

разработчиков по странам. Точного значения мы не знаем, набросаем случайные числа в csv-файл:

workers.csv:

```
country,salary,workers
USA,1700,130
China,1300,300
Russia,1450,200
Germany,1400,185
Hungary,1500,75
Ukraine,1300,68
France,1500,54
Sweden,1600,100
UK,1500,89
Spain,1400,45
```

Тогда код будет выглядеть так:

```
data = pd.read_csv('workers.csv')
df = data.pivot(index='salary', values='workers', columns='country')
df.head()
heatmap_plot = sns.heatmap(df, center=0, cmap='gist_ncar')
plt.show()
```

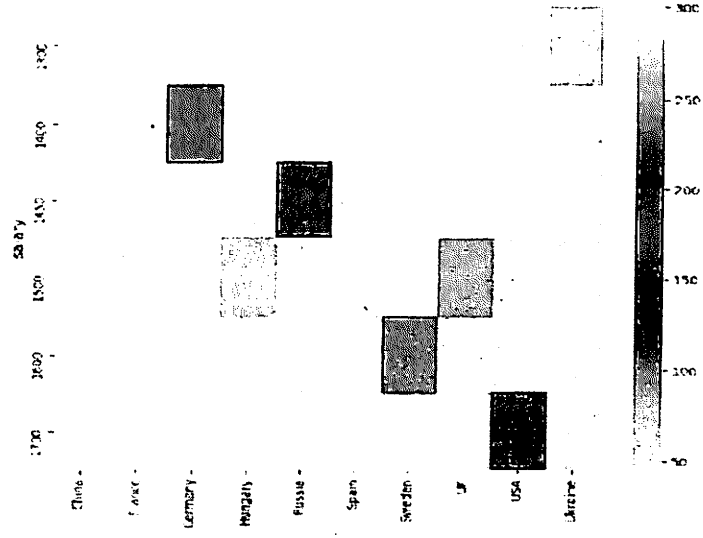
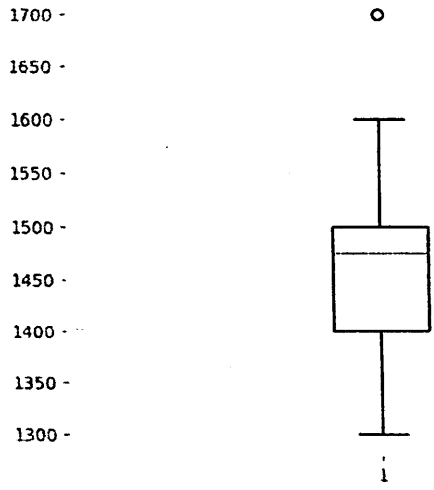


Диаграмма размаха ещё называется «ящик с усами» — смотрим среднее значение и «усами» показываем максимальные отклонения от него. Построимтак:

```
data= np.array(list(salary_2018.values())).astype(np.int)
plt.boxplot(data)
plt.show()
```



Радиальная диаграмма удобна, когда мы хотим сравнить параметры у одной сущности. Например, какие скиллы нужны веб-разработчику:

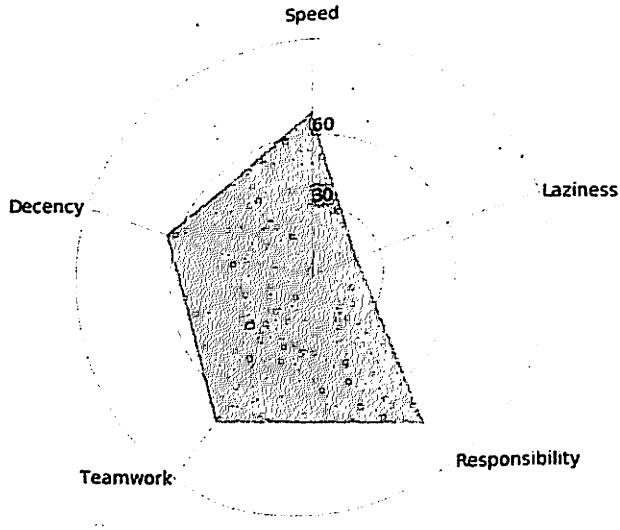
```
from math import pi
import matplotlib.pyplot as plt
# Вносим данные - какие скиллы хотим видеть в веб-разработчике
cat = ['Speed', 'Laziness', 'Responsibility', 'Teamwork', 'Decency']
values = [70, 20, 80, 70, 65]
N = len(cat)
x_as = [n / float(N) * 2 * pi for n in range(N)]
# Связываем последнее значение с первым чтобы построить радиальный график
values += values[:1]
x_as += x_as[:1]
# Устанавливаемцветитолщинулиний
plt.rc('axes', linewidth=0.5, edgecolor="#888888")
# Создаемдиаграмму
ax = plt.subplot(111, polar=True)
# Устанавливаемстилидлясетки
```

```
    ax.xaxis.grid(True, color="#888888", linestyle='solid', linewidth=0.5)
    ax.yaxis.grid(True, color="#888888", linestyle='solid', linewidth=0.5)
    ax.set_theta_offset(pi / 2)
    ax.set_theta_direction(-1)
    ax.set_rlabel_position(0)

    # Убираем стандартные метки
    plt.xticks(x_as[:-1], [])
    # Выводим шаг значения на график
    plt.yticks([30, 60], ["30", "60"])
    # Беремданныедлядиаграммы
    ax.plot(x_as, values, linewidth=0, linestyle='solid', zorder=3)
    # Заполняем область под значениями
    ax.fill(x_as, values, 'b', alpha=0.3)
    # Ограничиваем области
    plt.ylim(0, 100)
    # Отрисовываем все элементы
    for i in range(N):
      angle_rad = i / float(N) * 2 * pi

      if angle_rad == 0:
        ha, distance_ax = "center", 10
      elif 0 < angle_rad < pi:
        ha, distance_ax = "left", 1
      elif angle_rad == pi:
        ha, distance_ax = "center", 1
      else:
        ha, distance_ax = "right", 1

      ax.text(angle_rad, 100 + distance_ax, cat[i], size=10,
horizontalalignment=ha, verticalalignment="center")
    # Показываем итоговую диаграмму
    plt.show()
```


Speed
60
30
Laziness
Decency
Responsibility
Teamwork

TensorFlow — это мощная библиотека для создания нейронных сетей.

Как его установить? Сначала скачиваем whl файл, затем устанавливаем его, примерно вот так:

C:\Users\user\AppData\Local\Programs\Python\Python36\python.exe -m pip install tensorflow-1.2.0rc2-cp36-cp36m-win_amd64.whl

Итак, сначала попробуем создать простой нейрон с одним входом и выходом, который реализует линейную функцию:

```
import tensorflow as tf
#Создадим граф
graph = tf.get_default_graph()
#Создадим входное значение
input_value = tf.constant(1.0)
#Создадим переменную
weight = tf.Variable(0.8)
#Создадим выходное значение
output_value = weight * input_value
```

Мы создали простой нейрон со входом, выходом, и одним входным весовых коэффициентом. Чтобы попользоваться им, нам нужно создать сессию:

```
#создадим сессию
sess = tf.Session()
```

Затем инициализировать переменные:

```
init=tf.global_variables_initializer()
sess.run(init)
```

После этого мы можем вывести значение на экран:

```
print(sess.run(output_value))
```

У нас выведет 0.8 — результат умножения входного значения 1 на весовой коэффициент 0.8.

Теперь попробуем его обучить. Допустим, мы желаем получить на выходе не 0.8, а к примеру 2. В этом случае нам надо задать желаемое значение, функцию ошибки и оптимизатор:

```
#Создаем оптимизиатор
desired_output_value = tf.constant(2.0)
loss = (output_value - desired_output_value)**2 #Функция ошибки
optim = tf.train.GradientDescentOptimizer(learning_rate=0.025)
#Оптимизатор
grads_and_vars = optim.compute_gradients(loss)
```

А потом еще надо итерационно, много раз применить оптимизацию (вцикле):

```
#Обучаем
train_step = tf.train.GradientDescentOptimizer(0.025).minimize(loss)
for i in range(100):
  sess.run(train_step)
```

А вот весь код нашего примера:

```
import tensorflow as tf
#Создадим граф
graph = tf.get_default_graph()
#Создадим входное значение
input_value = tf.constant(1.0)
#Создадим переменную
weight = tf.Variable(0.8)
#Создадим выходное значение
output_value = weight * input_value
#создадим сессию
sess = tf.Session()
#Создаем оптимизиатор
desired_output_value = tf.constant(2.0)
loss = (output_value - desired_output_value)**2 #Функция ошибки
optim = tf.train.GradientDescentOptimizer(learning_rate=0.025)
#Оптимизатор
grads_and_vars = optim.compute_gradients(loss)
#Инициализируем переменные
init=tf.global_variables_initializer()
sess.run(init)
#Обучаем
train_step = tf.train.GradientDescentOptimizer(0.025).minimize(loss)
for i in range(100):
  sess.run(train_step)
```

```
print(sess.run(output_value))
```

В результате мы получим число, близкое к 2 (к нашему желаемому выходу), например 1.9929.

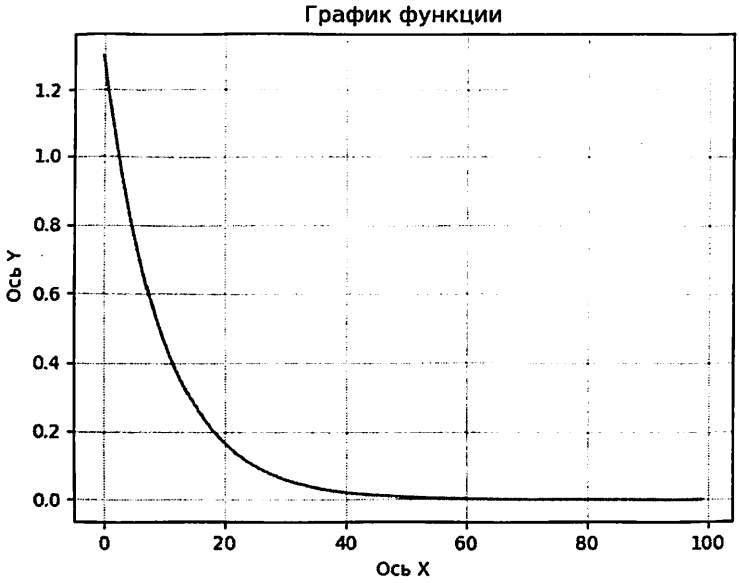
Часто для оцени процесса обучения смотрят график обучения, на котором, как правило, отображается ошибка обучения (как она уменьшается со временем).

Для этого нам надо импортируем саму библиотеку, в цикл обучения вставать код, который будет выводить значение ошибки на каждой итерации в список или массив, и отобразить этот список или массив на графике. Вот полный код программы с данными изменениями:

```
import tensorflow as tf
import matplotlib.pyplot as plt
#Создадим граф
graph = tf.get_default_graph()
#Создадим входное значение
input_value = tf.constant(1.0)
#Создадим переменную
weight = tf.Variable(0.8)
#Создаим выходное значение
output_value = weight * input_value
#создаимсессию
sess = tf.Session()
#Создаем оптимизиатор
desired_output_value = tf.constant(2.0)
loss = (output_value - desired_output_value)**2 #Функцияошибки
optim = tf.train.GradientDescentOptimizer(learning_rate=0.025)
#Оптимизатор
grads_and_vars = optim.compute_gradients(loss)
#Инициализируем переменные
init=tf.global_variables_initializer()
sess.run(init)
#задаем начальные массивы
x=[]
y=[]
#Обучаем
train_step = tf.train.GradientDescentOptimizer(0.025).minimize(loss)
for i in range(100):
  sess.run(train_step)
```

```
    x.append(i)
y.append(sess.run(loss))
print(sess.run(output_value))
#Строимграфик
fig = plt.figure()
plt.plot(x, y)
#Отображаем заголовки и подписи осей
plt.title('График функции')
plt.ylabel('Ось Y')
plt.xlabel('Ось X')
plt.grid(True)
plt.show()
```

А вот сам график обучения:



Одним из наиболее распространенных применений TensorFlow является распознавание и классификация изображений.

Если вы не понимаете основных концепций распознавания изображений, вам будет сложно полностью понять основную часть этой статьи. Поэтому, прежде чем мы продолжим, давайте определимся с терминологией.

TensorFlow - это библиотека с открытым исходным кодом, созданная для Python командой Google Brain. TensorFlow компилирует множество различных алгоритмов и моделей, позволяя пользователю реализовать глубокие нейронные сети для использования в таких задачах, как распознавание и классификация изображений, а также обработка естественного языка. TensorFlow - это мощный фреймворк, который функционирует путем реализации ряда узлов обработки, каждый из которых представляет математическую операцию, а весь ряд узлов называется «графом».

Распознавание изображения относится к задаче ввода изображения в нейронную сеть и присвоения какой-либо метки для этого изображения. Метка, которую выводит сеть, будет соответствовать заранее определенному классу. Может быть присвоено как сразу несколько классов, так и только один. Если существует всего только один класс, обычно применяется термин «распознавание», тогда как задача распознавания нескольких классов часто называется «классификацией».

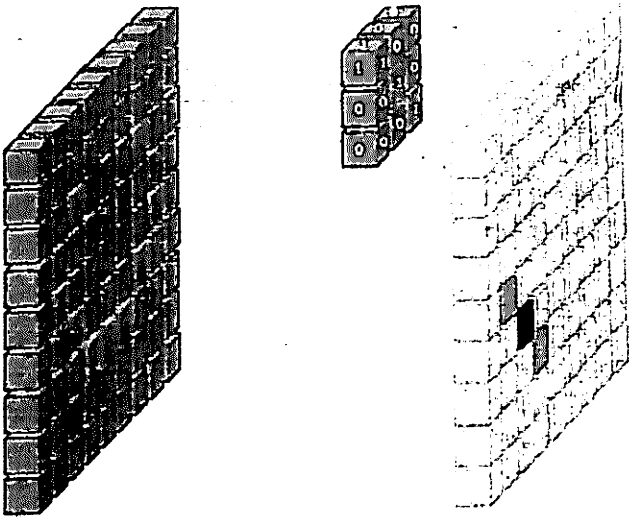
Подмножество классификаций изображений - является уже определением объектов, когда определенные экземпляры объектов идентифицируются как принадлежащие к определенному классу, например, животные, автомобили или люди.

Ярким примером такой классификации является решение самой распространённой капчи — ReCaptcha v2 от Google, где из набора картинок необходимо выбрать только те, которые принадлежат к указанному в описании классу.

Чтобы выполнить распознавание или классификацию изображений, нейронная сеть должна выполнить извлечение признаков. Признаки - это элементы данных, которые представляют максимальный интерес и которые будут передаваться по нейросети. В конкретном случае распознавания изображений такими признаками являются группы пикселей, такие как линии и точки, которые сеть будет анализировать на наличие паттерна.

Распознавание признаков (или извлечение признаков) - это процесс извлечения соответствующих признаков из входного изображения, чтобы их можно было проанализировать. Многие изображения содержат аннотации или метаданные, которые помогают нейросети находить соответствующие признаки.

Понимание о том, как нейронная сеть распознает изображения, поможет вам при реализации модели нейронной сети, поэтому давайте кратко рассмотрим процесс распознавания изображений в следующих разделах.



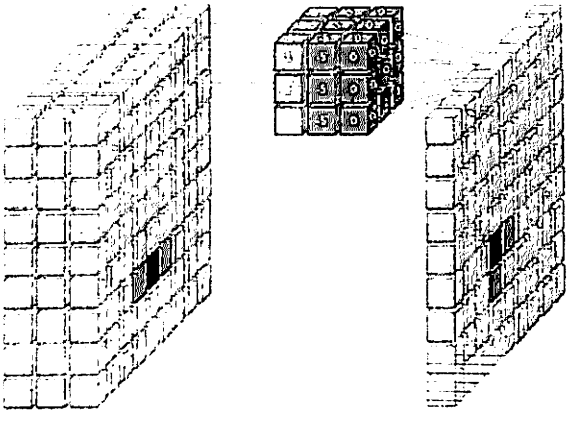
Первый слой нейронной сети принимает все пиксели в изображении. После того, как все данные введены в сеть, к изображению применяются различные фильтры, которые формируют понимание различных частей изображения. Это извлечение признаков, которое создает «карты признаков».

Этот процесс извлечения признаков из изображения выполняется с помощью «сверточного слоя», и свертка просто формирует представление части изображения. Именно из этой концепции свертки мы получаем термин "Сверточная нейронная сеть" (Convolutional Neural Network, CNN) — тип нейронной сети, наиболее часто используемый в классификации и распознавании изображений.

Если вы хотите визуализировать, как именно работает создание карт признаков, представьте себе процесс поднесения фонарика к изображению в темной комнате. Когда вы скользите лучом по картинке, вы узнаёте об особенностях изображения. Фильтр - это то, что сеть использует для формирования представления об изображении, и в этой метафоре свет от фонарика является фильтром.

Ширина луча вашего фонарика определяет размер фрагмента изображения, который вы просматриваете за один раз, и нейронные сети имеют аналогичный параметр — размер фильтра. Размер фильтра влияет на то, сколько пикселей проверяется за один раз. Общий размер фильтра, используемого в CNN, равен 3, и он охватывает как высоту, так и ширину, поэтому фильтр проверяет область пикселей 3 x 3.

Свёрточная сеть со слоями признаков



В то время как размер фильтра покрывает высоту и ширину фильтра, глубина фильтра также должна быть указана.

Дело в том, что цифровые изображения отображаются в виде высоты, ширины и некоторого значения RGB, которое определяет цвет пикселя, поэтому отслеживаемая «глубина» - это количество цветовых каналов, которые имеет изображение. Изображения в градациях серого (не цветные) имеют только 1 цветной канал, в то время как цветные изображения имеют глубину в 3 канала.

Все это означает, что для фильтра размером в 3, примененного к полноцветному изображению, итоговые размеры этого фильтра будут 3 x 3 x 3. Для каждого пикселя, охватываемого этим фильтром, сеть умножает значения фильтра на значения самих пикселей, чтобы получить числовое представление этого пикселя. Затем этот процесс выполняется для всего изображения, чтобы получить полное представление.

Фильтр перемещается по остальной части изображения в соответствии с параметром, называемым «шаг», который определяет, на сколько пикселей должен быть перемещен фильтр после того, как он вычислит значение в своей текущей позиции. Обычный размер шага для CNN - 2.

Конечным результатом всех этих расчетов является карта признаков. Этот процесс обычно выполняется с несколькими фильтрами, которые помогают сохранить сложность изображения.

После того, как карта признаков изображения была создана, значения, представляющие изображение, передаются через функцию активации или слой активации. Функция активации принимает эти значения, которые благодаря сверточному слою находятся в линейной форме (то есть просто список чисел) и увеличивает их нелинейность, поскольку сами изображения являются нелинейными.

Типичной функцией активации, используемой для достижения этой цели, является выпрямленная линейная единица (ReLU), хотя есть и некоторые другие функции активации, которые также иногда используются.

После активации данные отправляются через объединяющий слой. Объединение «упрощает» изображение: берёт информацию, которая представляет изображение, и сжимает её.

Процесс объединения в пул делает сеть более гибкой и способной лучше распознавать объекты и изображения на основе соответствующих функций.

Когда мы смотрим на изображение, нас, как правило, волнует не вся информация (например, что на заднем плане изображения), а только признаки, которые нас интересуют — люди, животные и т. д.

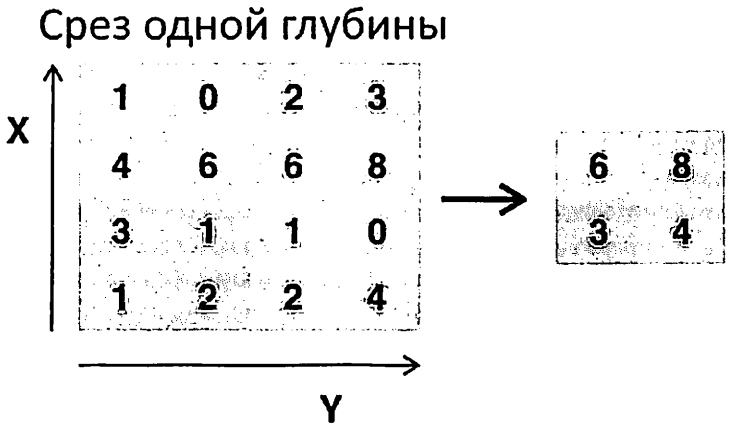
Аналогично, объединяющий слой в CNN избавится от ненужных частей изображения, оставив только те части, которые он считает релевантными, в зависимости от заданного размера объединяющего слоя.

Поскольку сеть должна принимать решения относительно наиболее важных частей изображения, расчёт идёт на то, что она изучит только те части изображения, которые действительно представляют суть рассматриваемого объекта.

Это помогает предотвратить "переобучение" — когда сеть слишком хорошо изучает все аспекты учебного примера и уже не может обобщать новые данные, поскольку учитывает нерелевантные отличия.

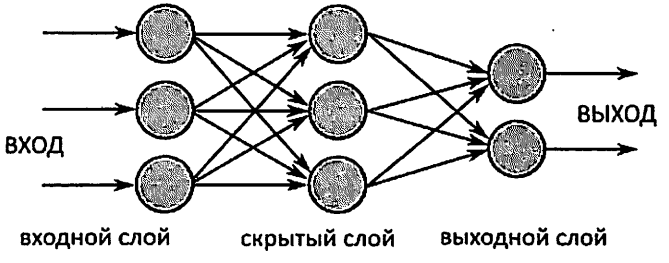
Существуют различные способы объединения значений, но чаще всего используется максимальное объединение. Максимальное объединение подразумевает взятие максимального значения среди пикселей в пределах одного фильтра (в пределах одного фрагмента изображения). Это отсеивает 3/4 информации, при условии использования фильтра размером 2 x 2.

Максимальные значения пикселей используются для того, чтобы учесть возможные искажения изображения, а количество параметров (размер изображения) уменьшены, чтобы контролировать переобучение. Существуют и другие принципы объединения, такие как среднее или суммарное объединение, но они используются не так часто, поскольку максимальное объединение даёт большую точность.



Последние слои нашей CNN — плотно связанные слои — требуют, чтобы данные были представлены в форме вектора для дальнейшей обработки. По этой причине данные необходимо «свести воедино». Для этого значения сжимаются в длинный вектор или столбец последовательно упорядоченных чисел.

Конечные слои CNN представляют собой плотно связанные слои или искусственную нейронную сеть (Artificial neural networks (ANN)). Основной функцией ANN является анализ входных признаков и объединение их в различные атрибуты, которые помогут в классификации. Эти слои образуют наборы нейронов, которые представляют различные части рассматриваемого объекта, а набор нейронов может представлять собой, например, висячие уши собаки или красноту яблока. Когда достаточное количество этих нейронов активируется в ответ на входное изображение, то оно будет классифицировано как объект.



Ошибка или разница между рассчитанными значениями и ожидаемым значением в обучающем наборе рассчитывается с помощью ANN. Затем сеть подвергается методу обратного распространения ошибки, где рассчитывается влияние данного нейрона на нейрон в следующем слое и затем его влияние (вес) корректируется. Это сделано для оптимизации производительности модели. Этот процесс повторяется снова и снова: так сеть обучается на данных и изучает связи между входными признаками и выходными классами.

Нейроны в средних полностью связанных слоях будут выводить двоичные значения, относящиеся к возможным классам. Если у вас есть четыре разных класса (скажем, собака, машина, дом и человек), нейрон будет иметь значение «1» для класса, который, как он считает, представляет изображение, и значение «0» для других классов.

Конечный полностью связанный слой, получив выходные данные предыдущего слоя, присваивает вероятность каждому из классов в пределах единицы (в совокупности). Если категории «собака» присвоено значение 0,75 — это означает 75% вероятность того, что изображение является собакой.

Прежде чем мы перейдем к примеру обучения классификатора изображений, давайте уделим немного времени пониманию рабочего процесса или "конвейера" машинного обучения. Процесс обучения модели нейронной сети является довольно стандартным и может быть поделен на четыре различных этапа.

Классификатор изображений теперь обучен и изображения могут быть переданы в CNN, которая теперь выведет предположение о содержании этого изображения.

Во-первых, вам нужно будет собрать свои данные и оформить их так, чтобы нейросеть смола обучаться на них. Это включает в себя сбор изображений и их маркировку. Даже если вы скачали набор данных, подготовленный кем-то другим, то скорее всего вам всё-равно потребуется предварительная обработка или подготовка, прежде чем вы сможете использовать его для обучения. Подготовка данных - это искусство само по себе, связанное с решением таких проблем, как пропущенные значения, поврежденные данные, данные в неправильном формате, неправильные метки и т. д.

Создание модели нейронной сети включает выбор различных параметров и гиперпараметров. Вы должны принять решение о количестве слоев, используемых в вашей модели, о том, каким будет размер входных и выходных слоев, какие функции активации вы будете использовать, будете ли вы использовать исключение (Dropout) и т. д.

Понимание того, какие параметры и гиперпараметры стоит использовать, придет со временем (изучать придётся много), но существуют некоторые базовые методы, которые вы можете использовать на старте, и мы рассмотрим некоторые из них в нашем примере.

После того, как ваша модель создана, вам просто остаётся создать экземпляр модели и подогнать его к своим данным для обучения. Наибольшее внимание при обучении модели уделяется количеству требуемого для обучения времени. Вы можете указать продолжительность обучения сети, задав количество эпох обучения. Чем дольше вы тренируете модель, тем выше ее эффективность, но если использовать слишком много эпох обучения - вы рискуете переобучить модель.

Выбор количества эпох для обучения - это то, что вы научитесь определять со временем, и, как правило, следует всегда сохранять веса нейросети между тренировочными сессиями, чтобы вам не нужно было начинать сначала после достижения определенного прогресса в обучении.

Существует несколько шагов для оценки модели. Первым шагом является сравнение производительности модели с набором проверочных данных: тех данных, на которых модель не была обучена. Таким образом, вы проверите работу модели с этим новым набором данных и проанализируете её эффективность с помощью различных показателей.

Существуют различные метрики для определения производительности модели нейронной сети, но наиболее распространенной является «точность», то есть количество правильно классифицированных изображений, делённое на общее количество изображений в вашем наборе данных.

После того, как вы увидите точность модели в проверочном наборе данных, вы, вероятно, снова вернетесь и до-обучите сеть, используя слегка подправленные параметры, поскольку вряд ли будете удовлетворены эффективностью своей сети при первой тренировке. Вы будете продолжать настраивать параметры своей сети, повторно обучать ее и измерять эффективность, пока не будете удовлетворены точностью сети.

Наконец, вы проверите эффективность сети на тестовом наборе. Это еще один набор данных, который ваша модель никогда не видела раньше.

Возможно, вам интересно: зачем нужен ещё один тестовый набор данных? Ведь вы уже получили представление о точности вашей модели, разве не это было целью “проверочного набора”?

Всё дело в том, что все изменения параметров, которые вы производили, донастраивая сеть при работе с “проверочным набором данных” в сочетании с многократным повторным тестированием этого набора — могли привести к тому, что ваша сеть изучила некоторые

особенности набора, но при этом она не будет так же хорошо обобщать данные вне выборки. Именно поэтому следует предоставить сети абсолютно новые тестовые данные.

Цель тестового набора - проверить наличие проблем, таких как переобучение, чтобы быть более уверенными в том, что ваша модель действительно пригодна для работы в реальном мире.

Мы уже многое рассмотрели и если вся эта информация была, возможно, немного неясной, то объединение вышеописанных концепций в выборочном классификаторе, обученном на наборе данных, должно окончательно все прояснить. Итак, давайте рассмотрим полный пример распознавания изображений с использованием Keras — от загрузки данных до оценки эффективности модели.

Для начала нам понадобится набор данных для обучения. В этом примере мы будем использовать известный набор данных CIFAR-10. CIFAR-10 - это большой набор данных, содержащий более 60000 изображений, представляющих 10 различных классов объектов, таких как кошки, самолеты и автомобили.

Изображения являются полноцветными RGB, но они достаточно малы, всего 32 x 32. Отличительной особенностью набора данных CIFAR-10

является то, что он поставляется в комплекте с Keras, поэтому загрузить набор данных очень просто, а сами изображения нуждаются лишь в минимальной предварительной обработке.

Первое, что мы должны сделать, это импортировать необходимые библиотеки. Вы ещё увидите, как именно этот импорт происходит по ходу дела, а пока же просто имейте в виду, что мы будем использовать Numpy и различные модули, связанные с Keras:

```
import numpy
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
```

Мы собираемся использовать случайный SEED (симметричный блочный криптоалгоритм на основе Сети Фейстеля), чтобы результаты, полученные в этой статье, могли быть воспроизведены вами, поэтому нам нужен numpy:

```
# Set random seed for purposes of reproducibility
seed = 21
```

Теперь нам нужно осуществить еще один импорт: сам набор данных.

```
from keras.datasets import cifar10
```

Теперь давайте загрузим набор данных. Мы можем сделать это просто указав, в какие переменные мы хотим загрузить данные, а затем использовать функцию load_data():

```
# loading in the data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

В большинстве случаев вам потребуется выполнить некоторую предварительную обработку ваших данных, чтобы подготовить их к использованию, но поскольку мы используем уже готовый и упакованный набор данных, то такая обработка сведена к минимуму. Одним из действий, которые мы хотим сделать, будет нормализация входных данных.

Если значения входных данных находятся в слишком широком диапазоне, это может отрицательно повлиять на работу сети. В нашем случае

входными значениями являются пиксели в изображении, которые имеют значение от 0 до 255.

Таким образом, чтобы нормализовать данные, мы можем просто разделить значения изображения на 255. Для этого нам сначала нужно перевести данные в формат с плавающей запятой, поскольку в настоящее время они являются целыми числами. Мы можем сделать это, используя Numpy команду astype(), а затем объявить желаемый тип данных:

```
# normalize the inputs from 0-255 to between 0 and 1 by dividing by
255
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Следующая вещь, которую нам нужно сделать, чтобы подготовить данные для сети, - перевести их в унитарный код. Не будем вдаваться в подробности унитарного кодирования, но знайте, что изображения не могут использоваться нейросетью в том виде, в каком они есть — их нужно сначала кодировать, а лучше всего при проведении двоичной классификации использовать именно унитарное кодирование.

Мы успешно применяем здесь двоичную классификацию, потому что изображение либо принадлежит одному определённому классу, либо нет: оно не может быть где-то посередине. Для унитарного кодирования используется команда Numpy to_categorical(). Вот почему мы импортировали функцию np_utils из Keras, так как она содержит to_categorical().

Нам также нужно задать количество классов в наборе данных, чтобы мы поняли, до скольких нейронов сжимать конечный слой:

```
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
class_num = y_test.shape[1]
```

Мы достигли стадии проектирования модели CNN. Первое, что нужно сделать, это определить формат, который мы хотели бы использовать для модели. У Keras есть несколько различных форматов (планов) для построения моделей, но наиболее часто используется Sequential - поэтому мы импортировали его из Keras.

```
model = Sequential()
```

Первый слой нашей модели - это сверточный слой. Он будет принимать входные данные и пропускать их через сверточные фильтры.

При реализации этого в Keras, мы должны указать количество каналов (фильтров), которое нам нужно (а это 32), размер фильтра (3 x 3 в нашем случае), форму входа (при создании первого слоя), функцию активации и отступы.

Как уже упоминалось, relu является наиболее распространенной функцией активации, а отступы мы определим через padding = 'same', то есть, мы не меняем размер изображения:

```
model.add(Conv2D(32, (3, 3), input_shape=X_train.shape[1:], padding='same'))
model.add(Activation('relu'))
```

Вы также можете объединить в одну строку нужные команды, например так:

```
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), activation='relu', padding='same'))
```

Теперь мы создадим исключающий слой для предотвращения переобучения, который случайным образом устраняет соединения между слоями (0,2 означает, что он отбрасывает 20% существующих соединений):

```
model.add(Dropout(0.2))
```

Также мы можем выполнить пакетную нормализацию. Пакетная нормализация нормализует входные данные, поступающие в следующий слой, гарантируя, что сеть всегда создает функции активации с тем же распределением, которое нам нужно:

```
model.add(BatchNormalization())
```

Теперь следует еще один сверточный слой, но размер фильтра увеличивается, так что сеть уже может изучать более сложные представления:

```
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
```

А вот и объединяющий слой, который, как обсуждалось ранее, помогает сделать классификатор изображений более корректным, чтобы он мог изучать релевантные шаблоны. Также опишем исключение (Dropout) и пакетную нормализацию:

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
```

Это основа рабочего процесса в первой части реализации CNN: свертка, активация, исключение, объединение. Теперь вы понимаете, зачем мы импортировали Dropout, BatchNormalization, Activation, Conv2d и MaxPooling2d.

Вы можете варьировать количество сверточных слоев по своему вкусу, но каждый из них увеличивает вычислительные затраты. Обратите внимание, что при добавлении сверточных слоев вы обычно увеличиваете и количество фильтров, чтобы модель могла выучить более сложные представления. Если числа, выбранные для этих слоев, кажутся несколько произвольными, то просто знайте, что рекомендуется увеличивать фильтры постепенно, устанавливая значение 2 в степени (2^n), что может дать небольшое преимущество при обучении модели на GPU.

Важно не иметь слишком много объединяющих уровней, так как каждый из них отбрасывает часть данных. Слишком частое объединение приведет к тому, что плотно связанные слои почти ничего не узнают, когда данные достигнут их.

Необходимое количество объединяющих слоев зависит от выполняемой задачи — это то, что вы определите со временем. Поскольку изображения в нашем наборе уже достаточно малы, мы не будем объединять их более двух раз.

Теперь вы можете повторить эти слои, чтобы дать вашей сети больше представлений для работы:

```
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
```

После того, как мы закончили со сверточными слоями, нам нужно сжать данные, поэтому мы импортировали функцию Flatten выше. Мы также добавим слой исключения снова:

```
model.add(Flatten())
model.add(Dropout(0.2))
```

Теперь мы используем импортированную функцию Dense и создаем первый плотно связанный слой. Нам нужно указать количество нейронов в плотном слое. Обратите внимание, что число нейронов в последующих слоях уменьшается, в конечном итоге приближаясь к тому же числу нейронов, что и классы в наборе данных (в данном случае 10). Ограничение ядра может упорядочить данные в процессе обучения, что также помогает предотвратить переобучение. Вот почему мы импортировали maxnorm ранее.

```
model.add(Dense(256, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(128, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
```

В этом последнем слое мы уравниваем количество классов с числом нейронов. Каждый нейрон представляет класс, поэтому на выходе этого слоя будет вектор из 10 нейронов, каждый из которых хранит некоторую вероятность того, что рассматриваемое изображение принадлежит его классу.

Наконец, функция активации softmax выбирает нейрон с наибольшей вероятностью в качестве своего выходного значения, предполагая, что изображение принадлежит именно этому классу:

```
model.add(Dense(class_num))
model.add(Activation('softmax'))
```

Теперь, когда мы разработали модель, которую хотим использовать, остаётся лишь скомпилировать ее. Давайте укажем количество эпох для обучения, а также оптимизатор, который мы хотим использовать.

Оптимизатор - это то, что настроит веса в вашей сети так, чтобы приблизиться к точке с наименьшими потерями. Алгоритм Адама является одним из наиболее часто используемых оптимизаторов, потому что он дает высокую производительность в большинстве задач:

```
epochs = 25
optimizer = 'adam'
```

Теперь скомпилируем модель с выбранными параметрами. Давайте также укажем метрику для оценки.

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

Мы также можем распечатать сводку по модели, чтобы получить представление о модели в целом.

```
print(model.summary())
```

Распечатка сводки даст нам некоторую информацию:

Results:

Layer (type) Output Shape Param #

conv2d_1 (Conv2D) (None, 32, 32, 32) 896

activation_1 (Activation) (None, 32, 32, 32) 0

dropout_1 (Dropout) (None, 32, 32, 32) 0

batch_normalization_1 (Batch (None, 32, 32, 32) 128

conv2d_2 (Conv2D) (None, 32, 32, 64) 18496

activation_2 (Activation) (None, 32, 32, 64) 0

max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64) 0

dropout_2 (Dropout) (None, 16, 16, 64) 0

batch_normalization_2 (Batch (None, 16, 16, 64) 256

conv2d_3 (Conv2D) (None, 16, 16, 64) 36928

activation_3 (Activation) (None, 16, 16, 64) 0

max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64) 0

dropout_3 (Dropout) (None, 8, 8, 64) 0

batch_normalization_3 (Batch (None, 8, 8, 64) 256

conv2d_4 (Conv2D) (None, 8, 8, 128) 73856

activation_4 (Activation) (None, 8, 8, 128) 0

dropout_4 (Dropout) (None, 8, 8, 128) 0

batch_normalization_4 (Batch (None, 8, 8, 128) 512

flatten_1 (Flatten) (None, 8192) 0

dropout_5 (Dropout) (None, 8192) 0

dense_1 (Dense) (None, 256) 2097408

activation_5 (Activation) (None, 256) 0

dropout_6 (Dropout) (None, 256) 0

batch_normalization_5 (Batch (None, 256) 1024

dense_2 (Dense) (None, 128) 32896

activation_6 (Activation) (None, 128) 0

dropout_7 (Dropout) (None, 128) 0

batch_normalization_6 (Batch (None, 128) 512
dense_3 (Dense) (None, 10) 1290
activation_7 (Activation) (None, 10) 0
Total params: 2,264,458
Trainable params: 2,263,114
Non-trainable params: 1,344

Теперь мы приступаем к обучению модели. Для этого нам нужно вызвать функцию fit () для модели и передать выбранные параметры.

Вот где используется SEED, выбранный в целях воспроизводимости.

```
numpy.random.seed(seed)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=64)
```

Возьмём тренировочный набор в 50000 образцов и проверочный в 10000 образцов.

Запуск этого куска кода даст

```
Epoch 1/25

64/50000 [..............................] - ETA: 16:57 - loss: 3.1479 - acc: 0.0938
128/50000 [..............................] - ETA: 10:12 - loss: 3.0212 - acc: 0.0938
192/50000 [..............................] - ETA: 7:57 - loss: 2.9781 - acc: 0.1250
256/50000 [..............................] - ETA: 6:48 - loss: 2.8830 - acc: 0.1484
320/50000 [..............................] - ETA: 6:07 - loss: 2.8878 - acc: 0.1469
384/50000 [..............................] - ETA: 5:40 - loss: 2.8732 - acc: 0.1458
448/50000 [..............................] - ETA: 5:20 - loss: 2.8842 - acc: 0.1406
...
...
...
49664/50000 [============================>.] - ETA: 1s - loss: 1.5160 - acc: 0.4611
49728/50000 [============================>.] - ETA: 1s - loss: 1.5157 - acc: 0.4612
49792/50000 [============================>.] - ETA: 1s - loss: 1.5153 - acc: 0.4614
49856/50000 [============================>.] - ETA: 0s - loss: 1.5147 - acc: 0.4615
49920/50000 [============================>.] - ETA: 0s - loss: 1.5144 - acc: 0.4617
49984/50000 [============================>.] - ETA: 0s - loss: 1.5141 - acc: 0.4617
50000/50000 [==============================] - 262s 5ms/step - loss: 1.5140 - acc: 0.4618 - val_loss: 1.0715 - val_acc: 0.6195
End of Epoch 1
```

Обратите внимание, что в большинстве случаев вам нужно иметь проверочный набор, отличный от набора для тестирования, поэтому вы должны указать процент данных обучения, которые будут использоваться в качестве набора для проверки. В этом случае мы просто передадим тестовые данные, чтобы убедиться, что тестовые данные отложены и не использовались для обучения. В этом примере мы будем иметь только тестовые данные, чтобы все было проще.

Теперь мы можем оценить модель и посмотреть, как она работает. Просто вызовите model.evaluate():

```
# Model evaluation
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

И вот мы получаем результат:

```
Accuracy: 83.01%
```

Теперь у нас есть обученная распознаванию изображений CNN. Неплохо для первого запуска, но вы, вероятно, захотите поэкспериментировать со структурой модели и параметрами, чтобы попытаться добиться лучшей эффективности.

Теоретические и экспериментальные работы по CNN закономерно подводят к вариантам использования нейросетей для решения практических повседневных задач. Наиболее актуальной в сфере распознавания и классификации изображений является задача по решению капчи, в частности — самой популярной на сегодняшний день Google ReCaptcha v2.

Несмотря на схожесть с нашим примером, на практике реализовать рабочую нейросеть для решения капчи представляется крайне затратным и неэффективным решением из-за постоянно меняющегося набора данных (картинок в капче). Столь частое и непредсказуемое обновление входящих данных влечёт за собой целую вереницу проблем:

- необходимость регулярно собирать и обрабатывать новые данные
- необходимость постоянного контроля процесса со стороны человека и внесение правок в модель по ходу работы (включая эксперименты с параметрами)
- необходимость мощного оборудования для обучения модели 24/7
- и т. д.

Универсальное решение проблемы обхода различных капч онлайн

Для решения капч в беспрерывном режиме, с высокой скоростью и сравнительно низкой стоимостью — большим спросом пользуются онлайн сервисы по распознаванию капч, которые привлекают для этого реальных пользователей.

На отечественном рынке лидером является сервис RuCaptcha.com, который выгодно отличается от конкурентов: * высокой точностью (до 99%) и скоростью решений (12 секунд для обычных текстовых капч и 24 секунды для ReCaptcha) * приемлемыми фиксированными ценами (цена не возрастает при увеличении нагрузки на сервера сервиса): 35 рублей за 1000 решений обычных капч и 160 рублей за 1000 решений ReCaptcha * возвратом средств за редкие неуспешные распознавания * технической возможностью решать огромные объёмы капч (более 10,000 в минуту) * простым и функциональным API * готовыми библиотеками и образцами кода для различных языков программирования * привлекательной партнёрской программой, позволяющей разработчикам и рефоводам получать до 15% от расходов привлечённых клиентов и 10% от доходов привлеченных в сервис работников.

Любые возникающие вопросы по работе сервиса — оперативно решаются службой поддержки через систему тикетов.

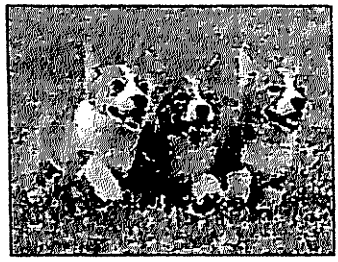
Свёрточные нейронные сети (CHC). Звучит как странное сочетание биологии и математики с примесью информатики, но как бы оно не звучало, эти сети - одни из самых влиятельных инноваций в области компьютерного зрения.

Впервые нейронные сети привлекли всеобщее внимание в 2012 году, когда Алекс Крижевски благодаря им выиграл конкурс ImageNet (грубо говоря, это ежегодная олимпиада по машинному зрению), снизив рекорд ошибок классификации с 26% до 15%, что тогда стало прорывом.

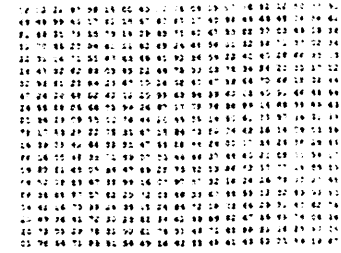
Сегодня глубинное обучения лежит в основе услуг многих компаний: Facebook использует нейронные сети для алгоритмов автоматического проставления тегов, Google - для поиска среди фотографий пользователя, Amazon - для генерации рекомендаций товаров, Pinterest - для персонализации домашней страницы пользователя, а Instagram - для поисковой инфраструктуры.

Но классический, и, возможно, самый популярный вариант использования сетей - это обработка изображений. Давайте посмотрим, как CHC используются для классификации изображений.

Задача классификации изображений - это приём начального изображения и вывод его класса (кошка, собака и т.д.) или группы вероятных классов, которая лучше всего характеризует изображение. Для людей это один из первых навыков, который они начинают осваивать с рождения.



What We See



What Computers See

Мы овладеваем им естественно, без усилий, став взрослыми. Даже не думая, мы можем быстро и легко распознать пространство, которое нас окружает, вместе с объектами. Когда мы видим изображение или просто смотрим на происходящее вокруг, чаще всего мы можем сразу охарактеризовать место действия, дать каждому объекту ярлык, и все это

происходит бессознательно, незаметно для разума. Эти навыки быстрого распознавания шаблонов, обобщения уже полученных знаний, и адаптации к различной запечатлённой на фото обстановке не доступны нашим электронным приятелям.

Когда компьютер видит изображение (принимает данные на вход), он видит массив пикселей. В зависимости от разрешения и размера изображения, например, размер массива может быть 32x32x3 (где 3 - это значения каналов RGB). Чтобы было понятней, давайте представим, у нас есть цветное изображение в формате JPG, и его размер 480x480. Соответствующий массив будет 480x480x3. Каждому из этих чисел присваивается значение от 0 до 255, которое описывает интенсивность пикселя в этой точке. Эти цифры, оставаясь бессмысленными для нас, когда мы определяем, что на изображении, являются единственными вводными данными, доступными компьютеру. Идея в том, что вы даете компьютеру эту матрицу, а он выводит числа, которые описывают вероятность класса изображения (.80 для кошки, .15 для собаки, .05 для птицы и т.д.).

Теперь, когда мы определили задачу, ввод и вывод, давайте думать о том, как подбираться к решению. Мы хотим, чтобы компьютер мог различать все данные ему изображения и распознавать уникальные особенности, которые делают собаку собакой, а кошку кошкой. У нас и этот процесс происходит подсознательно. Когда мы смотрим на изображение собаки, мы можем отнести его к конкретному классу, если у изображения есть характерные особенности, которые можно идентифицировать, такие как лапы или четыре ноги. Аналогичным образом компьютер может выполнять классификацию изображений через поиск характеристик базового уровня, например, границ и искривлений, а затем с помощью построения более абстрактных концепций через группы свёрточных слоев. Это общее описание того, что делают CHC. Теперь перейдём к специфике.

В начале немного истории. Когда вы впервые услышали термин сверточные нейронные сети, возможно подумали о чем-то связанном с нейронауками или биологией, и отчасти были правы. В каком-то смысле. CHC - это действительно прототип зрительной коры мозга. Зрительная кора имеет небольшие участки клеток, которые чувствительны к конкретным областям поля зрения.

Эту идею детально рассмотрели с помощью потрясающего эксперимента Хьюбел и Визель в 1962 году, в котором показали, что отдельные мозговые нервные клетки реагировали (или активировались) только при визуальном восприятии границ определенной ориентации. Например, некоторые нейроны активировались, когда воспринимали

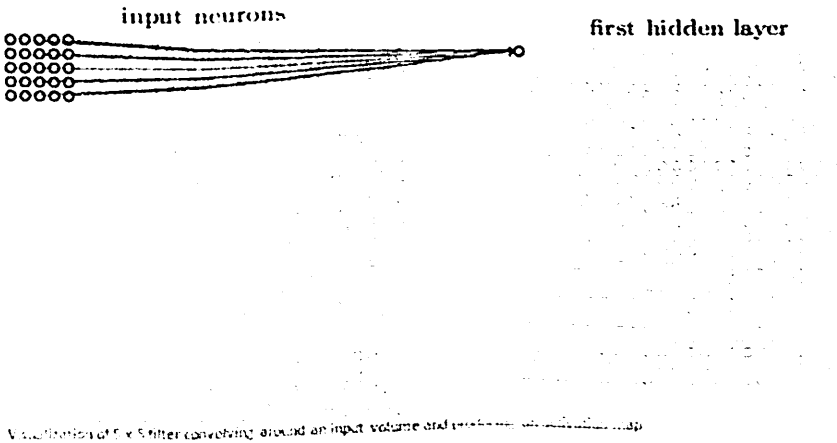
вертикальные границы, а некоторые - горизонтальные или диагональные. Хьюбел и Визель выяснили, что все эти нейроны сосредоточены в виде стержневой архитектуры и вместе формируют визуальное восприятие. Эту идею специализированных компонентов внутри системы, которые решают конкретные задачи (как клетки зрительной коры, которые ищут специфические характеристики) и используют машины, и эта идея - основа CHC.

Вернёмся к специфике. Что конкретно делают CHC? Берётся изображение, пропускается через серию свёрточных, нелинейных слоев, слоев объединения и полносвязных слоёв, и генерируется вывод. Как мы уже говорили, выводом может быть класс или вероятность классов, которые лучше всего описывают изображение. Сложный момент - понимание того, что делает каждый из этих слоев. Так что давайте перейдем к самому важному.

Первый слой в CHC всегда свёрточный. Вы же помните, какой ввод у этого свёрточного слоя? Как уже говорилось ранее, вводное изображение - это матрица 32 x 32 x 3 с пиксельными значениями. Легче всего понять, что такое свёрточный слой, если представить его в виде фонарика, который светит на верхнюю левую часть изображения. Допустим свет, который излучает этот фонарик, покрывает площадь 5 x 5. А теперь давайте представим, что фонарик движется по всем областям вводного изображения.

В терминах компьютерного обучения этот фонарик называется фильтром (иногда нейроном или ядром), а области, на которые он светит, называются рецептивным полем (полем восприятия). То есть наш фильтр - это матрица (такую матрицу ещё называют матрицей весов или матрицей параметров). Заметьте, что глубина у фильтра должна быть такой же, как и глубина вводного изображения (тогда есть гарантия математической верности), и размеры этого фильтра - 5 x 5 x 3. Теперь давайте за пример возьмем позицию, в которой находится фильтр. Пусть это будет левый верхний угол. Поскольку фильтр производит свёртку, то есть передвигается по вводному изображению, он умножает значения фильтра на исходные значения пикселей изображения (поэлементное умножение). Все эти умножения суммируются (всего 75 умножений). И в итоге получается одно число. Помните, оно просто символизирует нахождение фильтра в верхнем левом углу изображения. Теперь повторим этот процесс в каждой позиции. (Следующий шаг - перемещение фильтра вправо на единицу, затем еще на единицу вправо и так далее). Каждая уникальная позиция введённого изображения производит число. После прохождения фильтра по всем позициям получается матрица 28 x 28 x 1, которую называют функцией

активации или картой признаков. Матрица 28 x 28 получается потому, что есть 784 различных позиции, которые могут пройти через фильтр 5 x 5 изображения 32 x 32. Эти 784 числа преобразуются в матрицу 28 x 28.



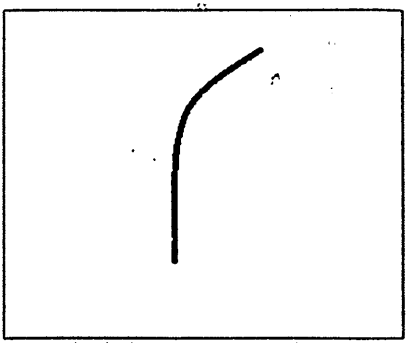
Допустим, теперь мы используем два 5 x 5 x 3 фильтра вместо одного. Тогда выходным значением будет 28 x 28 x 2.

Давайте поговорим о том, что эта свертка на самом деле делает на высоком уровне. Каждый фильтр можно рассматривать как идентификатор свойства. Когда я говорю свойство, я имею в виду прямые границы, простые цвета и кривые. Подумайте о самых простых характеристиках, которые имеют все изображения в общем.

Скажем, наш первый фильтр 7 x 7 x 3, и он будет детектором кривых. (Сейчас давайте игнорировать тот факт, что у фильтра глубина 3, и рассмотрим только верхний слой фильтра и изображения, для простоты). У фильтра пиксельная структура, в которой численные значения выше вдоль области, определяющей форму кривой (помните, фильтры, о которых мы говорим, это просто числа!).

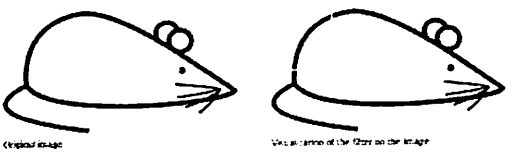
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

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



Помните, всё что нам нужно, это умножить значения фильтра на исходные значения пикселей изображения.

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

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

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

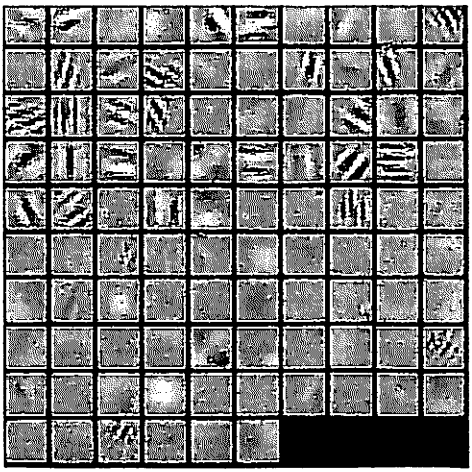
Visualization of the filter on the image Pixel representation of receptive field Pixel representation of filter

Multiplication and Summation = 0

В новой области изображения нет ничего, что фильтр определения кривой мог засечь. Помните, что вывод этого свёрточного слоя - карта свойств. В самом простом случае, при наличии одного фильтра свертки (и если этот фильтр - детектор кривой), карта свойств покажет области, в которых больше вероятности наличия кривых.

В этом примере в левом верхнем углу значение нашей 28 x 28 x 1 карты свойств будет 6600. Это высокое значение показывает, что, возможно, что-то похожее на кривую присутствует на изображении, и такая вероятность активировала фильтр. В правом верхнем углу значение у карты свойств будет 0, потому что на картинке не было ничего, что могло активировать фильтр.

Помните, что это только для одного фильтра. Это фильтр, который обнаруживает линии с изгибом наружу. Могут быть другие фильтры для линий, изогнутых внутрь или просто прямых. Чем больше фильтров, тем больше глубина карты свойств, и тем больше информации мы имеем о вводной картинке.



Visualizations of filters

Сегодня в традиционной сверточной нейронной сетевой архитектуре существуют и другие слои, которые перемежаются со свёрточными. Я очень рекомендую тем, кто интересуется темой, почитать об этих слоях, чтобы понять их функциональность и эффекты. Классическая архитектура CHC будет выглядеть так:

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected

Последний слой, хоть и находится в конце, один из важных — мы перейдём к нему позже. Давайте подытожим то, в чём мы уже разобрались. Мы говорили о том, что умеют определять фильтры первого свёрточного слоя. Они обнаруживают свойства базового уровня, такие как границы и кривые. Как можно себе представить, чтобы предположить какой тип объекта изображён на картинке, нам нужна сеть, способная распознавать свойства более высокого уровня, как например руки, лапы или уши.

Так что давайте подумаем, как выглядит выходной результат сети после первого свёрточного слоя. Его размер 28 x 28 x 3 (при условии, что мы используем три фильтра 5 x 5 x 3). Когда картинка проходит через один свёрточный слой, выход первого слоя становится вводным значением 2-го слоя. Теперь это немного сложнее визуализировать.

Когда мы говорили о первом слое, вводом были только данные исходного изображения. Но когда мы перешли ко 2-му слою, вводным

значением для него стала одна или несколько карт свойств - результат обработки предыдущим слоем. Каждый набор вводных данных описывает позиции, где на исходном изображении встречаются определенные базовые признаки.

Теперь, когда вы применяете набор фильтров поверх этого (пропускаете картинку через второй свёрточный слой), на выходе будут активированы фильтры, которые представляют свойства более высокого уровня. Типами этих свойств могут быть полукольца (комбинация прямой границы с изгибом) или квадратов (сочетание нескольких прямых ребер). Чем больше свёрточных слоёв проходит изображение и чем дальше оно движется по сети, тем более сложные характеристики выводятся в картах активации.

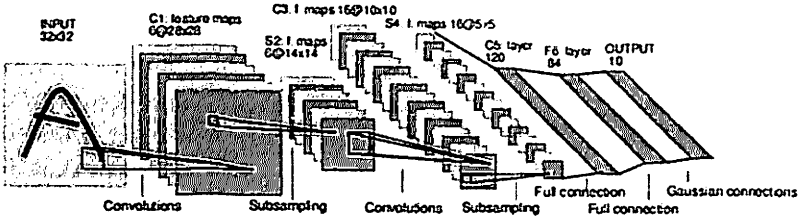
В конце сети могут быть фильтры, которые активируются при наличии рукописного текста на изображении, при наличии розовых объектов и т.д. Если вы хотите узнать больше о фильтрах в свёрточных сетях, Мэтт Зейлер и Роб Фергюс написали отличную научно-исследовательскую работу на эту тему. Ещё на ютубе есть видео Джейсона Йосински с отличным визуальным представлением этих процессов.

Еще один интересный момент. Когда вы двигаетесь вглубь сети, фильтры работают со все большим полем восприятия, а значит, они в состоянии обрабатывать информацию с большей площади первоначального изображения (простыми словами, они лучше приспособляются к обработке большей области пиксельного пространства).

Теперь, когда мы можем обнаружить высокоуровневые свойства, самое крутое - это прикрепление полносвязного слоя в конце сети. Этот слой берёт вводные данные и выводит N-пространственный вектор, где N - число классов, из которых программа выбирает нужный. Например, если вы хотите программу по распознаванию цифр, у N будет значение 10, потому что цифр 10. Каждое число в этом N-пространственном векторе представляет собой вероятность конкретного класса. Например, если результирующий вектор для программы распознавания цифр это [0 0,1 0,1 0,75 0 0 0 0 0,05], значит существует 10% вероятность, что на изображении "1", 10% вероятность, что на изображение "2", 75% вероятность - "3", и 5% вероятность - "9" (конечно, есть и другие способы представить вывод).

Способ, с помощью которого работает полносвязный слой - это обращение к выходу предыдущего слоя (который, как мы помним, должен выводить высокоуровневые карты свойств) и определение свойств, которые больше связаны с определенным классом. Например, если программа предсказывает, что на каком-то изображении собака, у карт свойств, которые

отражают высокоуровневые характеристики, такие как лапы или 4 ноги, должны быть высокие значения. Точно так же, если программа распознаёт, что на изображении птица - у неё будут высокие значения в картах свойств, представленных высокоуровневыми характеристиками вроде крыльев или клюва. Полносвязный слой смотрит на то, что функции высокого уровня сильно связаны с определенным классом и имеют определенные веса, так что, когда вы вычисляете произведения весов с предыдущим слоем, то получаете правильные вероятности для различных классов.



A Full Convolutional Neural Network (LeNet)

Это один из аспектов нейронных сетей, о котором я специально до сих пор не упоминал. Вероятно, это самая важная часть. Возможно, у вас появилось множество вопросов. Откуда фильтры первого свёрточного слоя знают, что нужно искать границы и кривые? Откуда полносвязный слой знает, что ищет карта свойств? Откуда фильтры каждого слоя знают, какие хранить значения? Способ, которым компьютер способен корректировать значения фильтра (или весов) - это обучающий процесс, который называют методом обратного распространения ошибки.

Перед тем, как перейти к объяснению этого метода, поговорим о том, что нейронной сети нужно для работы. Когда мы рождаемся, наши головы пусты. Мы не понимаем как распознать кошку, собаку или птицу. Ситуация с СНС похожа: до момента построения сети, веса или значения фильтра случайны. Фильтры не умеют искать границы и кривые. Фильтры верхних слоёв не умеют искать лапы и клювы.

Когда мы становимся старше, родители и учителя показывают нам разные картинки и изображения и присваивают им соответствующие ярлыки. Та же идея показа картинки и присваивания ярлыка используется в обучающем процессе, который проходит СНС. Давайте представим, что у нас есть набор обучающих картинок, в котором тысячи изображений собак, кошек и птиц. У каждого изображения есть ярлык с названием животного.

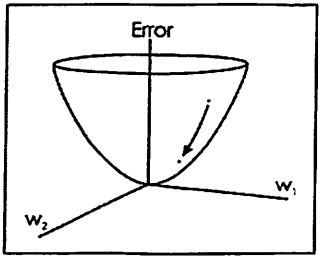
Метод обратного распространения ошибки можно разделить на 4 отдельных блока: прямое распространение, функцию потери, обратное распространение и обновление веса. Во время прямого распространения, берётся тренировочное изображение - как помните, это матрица 32 x 32 x 3 - и пропускается через всю сеть. В первом обучающем примере, так как все веса или значения фильтра были инициализированы случайным образом, выходным значением будет что-то вроде [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], то есть такое значение, которое не даст предпочтения какому-то определённому числу. Сеть с такими весами не может найти свойства базового уровня и не может обоснованно определить класс изображения. Это ведёт к функции потери.

Помните, то, что мы используем сейчас - это обучающие данные. У таких данных есть и изображение и ярлык. Допустим, первое обучающее изображение - это цифра 3. Ярлыком изображения будет [0 0 0 1 0 0 0 0 0 0]. Функция потери может быть выражена по-разному, но часто используется СКО (среднеквадратическая ошибка), это 1/2 умножить на (реальность - предсказание) в квадрате.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Примем это значение за переменную L. Как вы догадываетесь, потеря будет очень высокой для первых двух обучающих изображений. Теперь давайте подумаем об этом интуитивно. Мы хотим добиться того, чтобы спрогнозированный ярлык (вывод свёрточного слоя) был таким же, как ярлык обучающего изображения (это значит, что сеть сделала верное предположение).

Чтобы такого добиться, нам нужно свести к минимуму количество потерь, которое у нас есть. Визуализируя это как задачу оптимизации из математического анализа, нам нужно выяснить, какие входы (веса, в нашем случае) самым непосредственным образом способствовали потерям (или ошибкам) сети.



One way of visualizing the idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in a bowl shaped object. To do this, we have to take a derivative of the loss (visual terms, calculate the slope in every direction) with respect to the weights

(Один из способов визуализировать идею минимизации потери - это трёхмерный график, где веса нейронной сети (очевидно их больше, чем 2, но тут пример упрощен) это независимые переменные, а зависимая переменная - это потеря. Задача минимизации потерь - отрегулировать веса так, чтобы снизить потерю. Визуально нам нужно приблизиться к самой нижней точке чашеподобного объекта. Чтобы добиться этого, нужно найти производную потери (в рамках нарисованного графика - рассчитать угловой коэффициент в каждом направлении) с учётом весов).

Это математический эквивалент dL/dW, где W - веса определенного слоя. Теперь нам нужно выполнить обратное распространение через сеть, которое определяет, какие веса оказали большее влияние на потери, и найти способы, как их настроить, чтобы уменьшить потери. После того, как мы вычислим производную, перейдём к последнему этапу - обновлению весов. Возьмём все фильтровые веса и обновим их так, чтобы они менялись в направлении градиента.

Скорость обучения - это параметр, который выбирается программистом. Высокая скорость обучения означает, что в обновлениях веса делались более крупные шаги, поэтому образцу может потребоваться меньше времени, чтобы набрать оптимальный набор весов. Но слишком высокая скорость обучения может привести к очень крупным и недостаточно точным скачкам, которые помешают достижению оптимальных показателей.

Процесс прямого распространения, функцию потери, обратное распространение и обновление весов, обычно называют одним периодом дискретизации (или epoch - эпохой). Программа будет повторять этот процесс фиксированное количество периодов для каждого тренировочного изображения. После того, как обновление параметров завершится на последнем тренировочном образце, сеть в теории должна быть достаточно хорошо обучена и веса слоёв настроены правильно.

Сверточная нейронная сеть (с аббревиатурами CNN или ConvNets) представляет собой конкретный случай нейронных сетей глубокого обучения, которые уже использовались в конце 90-х годов, но которые в последние годы стали чрезвычайно популярными при достижении очень впечатляющих результатов в распознавании изображение, глубоко влияющее на область компьютерного зрения.

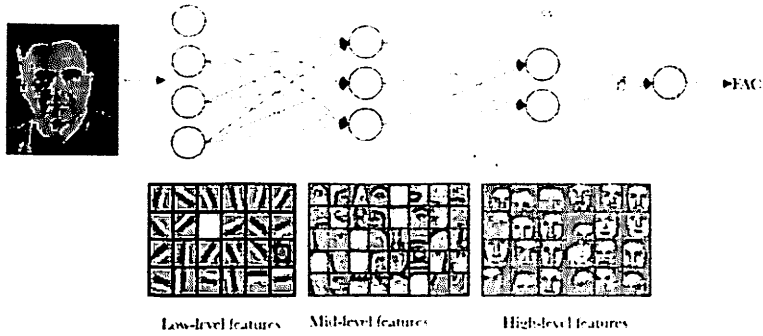
Сверточные нейронные сети очень похожи на нейронные сети: они образованы нейронами, которые имеют параметры в виде весов и смещений, которые могут быть изучены. Но отличительной особенностью CNN является то, что они делают явное предположение, что записи являются изображениями, что позволяет нам кодировать определенные свойства в архитектуре для распознавания определенных элементов в изображениях.

Чтобы получить интуитивное представление о том, как работают эти нейронные сети, давайте подумаем о том, как мы распознаем вещи. Например, если мы видим лицо, мы узнаем его, потому что у него есть уши, глаза, нос, волосы и т. Д. Затем, чтобы решить, является ли что-то лицо, мы делаем это, как если бы у нас были какие-то ментальные рамки проверки характеристики, которые мы отмечаем.

Иногда лицо может не иметь уха, потому что оно покрыто волосами, но мы также с определенной вероятностью классифицируем его как лицо, потому что мы видим глаза, нос и рот. На самом деле, мы можем рассматривать его как классификатор, которая предсказывает вероятность того, что входное изображение будет лицом или не будет лицом.

Но на самом деле, мы должны сначала узнать, что такое ухо или нос, чтобы знать, находятся ли они на изображении; то есть мы должны предварительно идентифицировать линии, края, текстуры или формы, которые похожи на те, которые содержат уши или носы, которые мы видели ранее. И это то, что поручено делать слоям сверточной нейронной сети.

Но идентификации этих элементов недостаточно, чтобы сказать, что что-то является лицом. Мы также должны быть в состоянии определить, как части лица соответствуют друг другу, относительные размеры и т. д.; иначе лицо не будет похоже на то, к чему мы привыкли.



Идея, которую мы хотим привести на этом наглядном примере, заключается в том, что в действительности в сверточной нейронной сети каждый уровень изучает разные уровни абстракции. Читатель может представить, что в сетях со многими уровнями можно определить более сложные структуры во входных данных.

Теперь, когда у нас есть интуитивное представление о том, как сверточные нейронные сети классифицируют изображение, мы представим пример распознавания цифр MNIST и из него мы представим два уровня, которые определяют сверточные сети, которые могут быть выражены в виде групп специализированных нейронов в двух операции: свертка и объединение.

Принципиальное различие между плотно связанным слоем и специализированным слоем в операции свертки, которую мы будем называть сверточным слоем, состоит в том, что плотный слой изучает глобальные шаблоны в своем глобальном входном пространстве, в то время как сверточные слои изучают локальные шаблоны в небольших окнах. два измерения.

Интуитивно понятным способом можно сказать, что основная цель сверточного слоя состоит в обнаружении элементов или визуальных элементов на изображениях, таких как края, линии, цветовые капли и т. Д. Это очень интересное свойство, поскольку после того, как оно изучило характеристику в определенном месте изображения он может распознать его позже в любой его части. Вместо этого в плотно связанной нейронной сети он должен снова изучить шаблон, если он появляется в новом месте изображения.

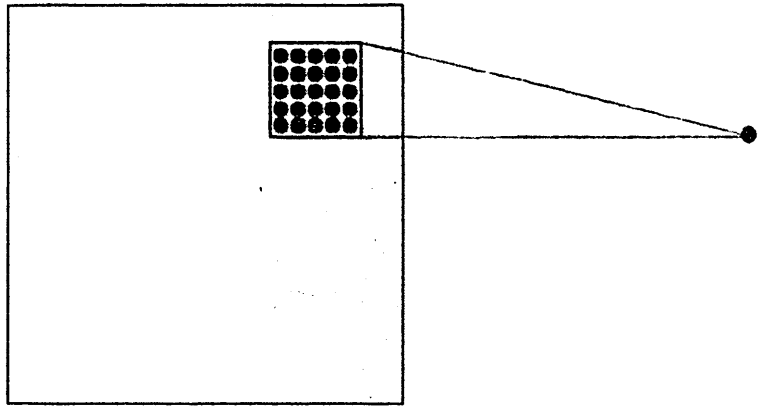
Другой важной особенностью является то, что сверточные слои могут изучать пространственные иерархии шаблонов, сохраняя пространственные

отношения. Например, первый сверточный уровень может изучать базовые элементы, такие как ребра, а второй сверточный уровень может изучать шаблоны, состоящие из базовых элементов, изученных на предыдущем уровне. И так до тех пор, пока не выучат очень сложные паттерны. Это позволяет сверточным нейронным сетям эффективно изучать все более сложные и абстрактные визуальные концепции.

Как правило, слои сверток работают на трехмерных тензорах, называемых картами объектов, с двумя пространственными осями высоты и ширины, а также осью канала, также называемой глубиной. Для цветного изображения RGB размер оси глубины равен 3, поскольку изображение имеет три канала: красный, зеленый и синий. Для черно-белого изображения, такого как цифры MNIST, размер оси глубины равен 1 (уровень серого).

В случае MNIST, в качестве входных данных для нашей нейронной сети мы можем представить пространство двумерных нейронов 28 × 28 (высота = 28, ширина = 28, глубина = 1). Первый слой скрытых нейронов, соединенных с нейронами входного слоя, который мы обсуждали, будет выполнять сверточные операции, которые мы только что описали. Но, как мы объяснили, не все входные нейроны связаны со всеми нейронами этого первого уровня скрытых нейронов, как в случае плотно связанных нейронных сетей: это делается только небольшими локализованными областями пространства входных нейронов, которые хранят пиксели изображения.

Объясненное визуально можно представить в виде:

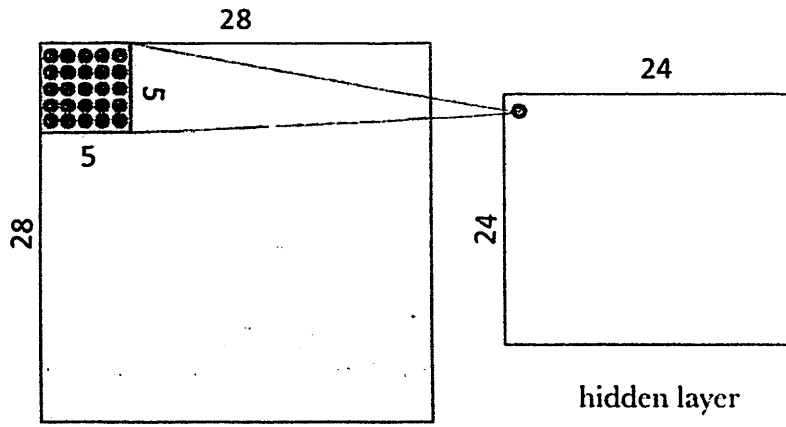


В нашем предыдущем примере каждый нейрон нашего скрытого слоя будет связан с небольшой областью 5 × 5 нейронов (то есть 25 нейронов)

входного слоя (28 × 28). Интуитивно понятно, что мы можем представить окно размером 5 × 5, которое скользит по всему входному слою нейронов 28 × 28, содержащему изображение. Для каждой позиции окна в скрытом слое есть нейрон, который обрабатывает эту информацию.

Визуально мы начинаем с окна в верхнем левом углу изображения, и это дает необходимую информацию для первого нейрона скрытого слоя. Затем мы сдвигаем окно на одну позицию вправо, чтобы «соединить» нейроны 5 × 5 входного слоя, включенного в это окно, со вторым нейроном скрытого слоя. И так, последовательно, мы проходим все пространство входного слоя, слева направо и сверху вниз.

Немного проанализировав конкретный случай, который мы предложили, отметим, что, если у нас есть вход 28 × 28 пикселей и окно 5 × 5, это определяет пространство 24 × 24 нейронов в первом скрытом слое, потому что мы можем Переместите только окно 23 нейронов вправо и 23 нейрона вниз, прежде чем ударить по правой (или нижней) границе входного изображения.



Мы хотели бы указать читателю, что сделанное нами допущение состоит в том, что окно перемещается вперед на 1 пиксель, как по горизонтали, так и по вертикали, когда начинается новая строка. Поэтому на каждом шаге новое окно перекрывает предыдущее, за исключением этой линии пикселей, которую мы выдвинули. Но, как мы увидим в следующем разделе, в сверточных нейронных сетях можно использовать различные длины шагов продвижения (параметр, называемый шаг). В сверточных

нейронных сетях вы также можете применить технику заполнения нулей по краям изображения, чтобы улучшить развертку, выполняемую с помощью скользящего окна. Параметр, определяющий эту технику, называется «заполнением», который мы также представим более подробно в следующем разделе, с помощью которого вы можете указать размер этого заполнения.

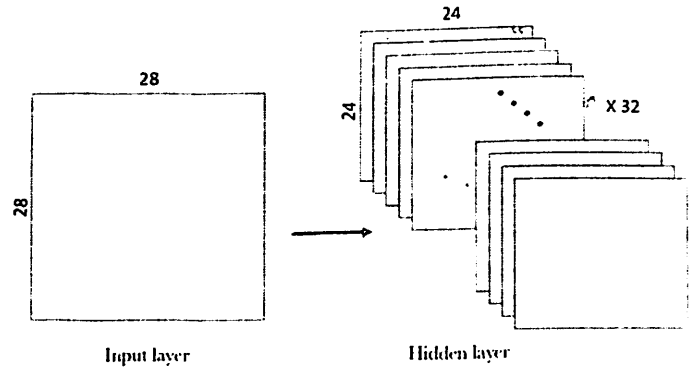
В нашем случае изучения и в соответствии с ранее представленным формализмом, чтобы «соединить» каждый нейрон скрытого слоя с 25 соответствующими нейронами входного слоя, мы будем использовать значение смещения $б$ и W весовая матрица размером 5 × 5, которую мы будем называть фильтром (или ядром). Значение каждой точки скрытого слоя соответствует скалярному произведению между фильтром и несколькими 25 нейронами (5 × 5) входного слоя.

Тем не менее, особая и очень важная вещь в сверточных сетях заключается в том, что мы используем один и тот же фильтр (тот же W матрица весов и того же $б$ смещение) для всех нейронов в скрытом слое: в нашем случае для 24 ⁄ 24 нейронов (всего 576 нейронов) первого слоя. Читатель может увидеть в этом конкретном случае, что это совместное использование радикально уменьшает количество параметров, которые были бы у нейронной сети, если бы мы этого не делали: оно исходит из 14 400 параметров, которые должны быть скорректированы (5 × 5 × 24 × 24). до 25 (5 × 5) параметров плюс смещения $б$.

Этот общий W матрица вместе с $б$ Смещение, которое мы уже говорили, мы называем фильтром в этом контексте сверточных сетей, аналогично фильтрам, которые мы используем для ретуширования изображений, которые в нашем случае используются для поиска локальных характеристик в небольших группах записей.

Но фильтр определяется матрицей W и уклон $б$ позволяет только определить конкретную характеристику на изображении; поэтому для распознавания изображений предлагается использовать несколько фильтров одновременно, по одному для каждой характеристики, которую мы хотим обнаружить. Вот почему полный сверточный слой в сверточной нейронной сети включает в себя несколько фильтров

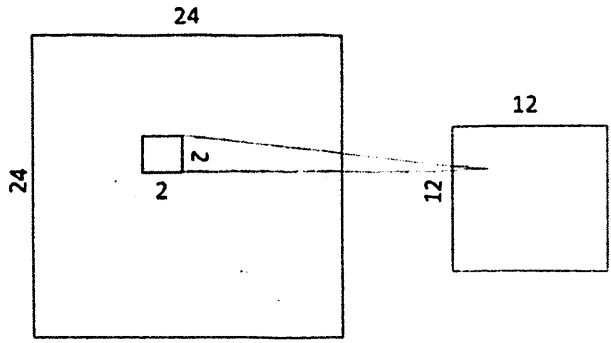
Обычный способ визуального представления этого сверточного слоя показан на следующем рисунке, где уровень скрытых слоев состоит из нескольких фильтров. В нашем примере мы предлагаем 32 фильтра, где каждый фильтр определен с W матрица 5 × 5 и уклон $б$,



В этом примере первый сверточный слой принимает входной тензор размера (28. 28, 1) и генерирует выходной размер (24. 24. 32), трехмерный тензор, содержащий 32 выходных значения 24 × 24 пикселя, результата вычисления 32 фильтры на входе.

В дополнение к сверточным слоям, которые мы только что описали, сверточные нейронные сети сопровождают сверточный слой пулами. которые обычно применяются сразу после сверточных слоев. Первый подход к пониманию того, для чего предназначены эти уровни, состоит в том, чтобы увидеть, что объединяющие слои упрощают информацию, собираемую сверточным слоем, и создают сжатую версию информации. содержащейся в них.

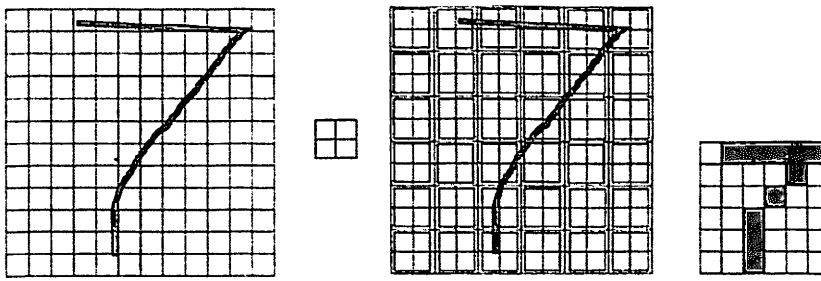
В нашем примере MNIST мы собираемся выбрать окно 2 × 2 сверточного слоя, и мы собираемся синтезировать информацию в точке на уровне пула. Визуально это можно выразить следующим образом:



Существует несколько способов сжатия информации, но обычный способ, который мы будем использовать в нашем примере, называется max-pooling, который в качестве значения сохраняет максимальное значение из тех, которые были в окне ввода 2 × 2 в нашем кейс. В этом случае мы делим на 4 размер вывода пула, оставляя изображение 12 × 12.

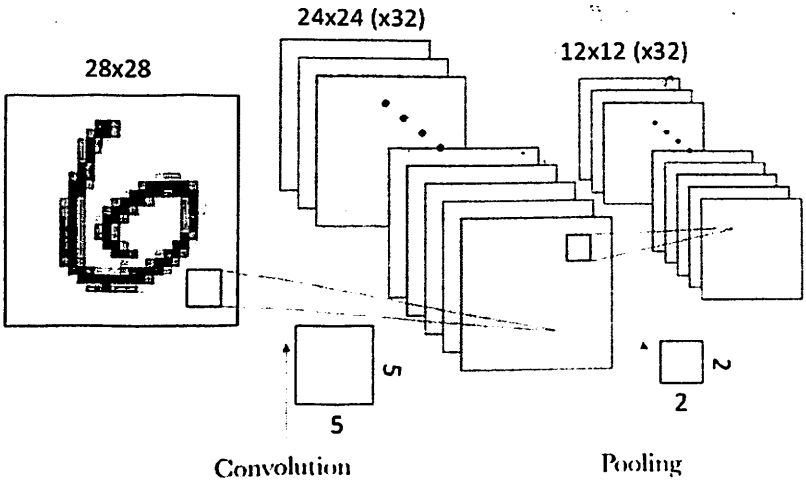
Среднее объединение может также использоваться вместо максимального объединения, где каждая группа точек входа преобразуется в среднее значение группы точек вместо ее максимального значения. Но в целом max-pooling работает лучше, чем альтернативные решения.

Интересно отметить, что с преобразованием пула мы поддерживаем пространственные отношения. Чтобы увидеть это наглядно, возьмите следующий пример матрицы 12 × 12, где мы представили «7» (давайте представим, что пиксели, которые мы передаем, содержат 1, а остальные 0; мы не добавили его в чертеж, чтобы упростить Это). Если мы применяем операцию максимального пула с окном 2 × 2 (мы представляем его в центральной матрице, которая делит пространство в мозаике с областями размера окна), мы получаем матрицу 6 × 6, где эквивалентное представление из 7 сохраняется (на рисунке справа, где нули отмечены белым, а точки со значением 1 черным):



Как упоминалось выше, сверточный уровень содержит более одного фильтра и, следовательно, поскольку мы применяем максимальное объединение к каждому из них в отдельности, уровень объединения будет содержать столько фильтров объединения, сколько существует сверточных

фильтров:



В результате, поскольку у нас было пространство 24 × 24 нейронов в каждом сверточном фильтре, после выполнения объединения мы имеем 12 × 12 нейронов, что соответствует 12 × 12 областям (размером 2 × 2 каждой области), которые появляются при делении пространство фильтра.

Давайте посмотрим, как этот пример сверточной нейронной сети можно запрограммировать с помощью Keras. Как мы уже говорили, для параметризации этапов свертки и объединения необходимо указать несколько значений. В нашем случае мы будем использовать упрощенную модель с шагом 1 в каждом измерении (размер шага, с которым скользит окно) и отступом 0 (заполнение нулями вокруг изображения). Оба гиперпараметра будут представлены ниже. Пул будет максимальным, как описано выше, с окном 2 × 2.

Давайте перейдем к реализации нашей первой сверточной нейронной сети, которая будет состоять из свертки, за которой следует максимальное объединение. В нашем случае у нас будет 32 фильтра, использующих окно 5 × 5 для сверточного слоя и окно 2 × 2 для пула. Мы будем использовать функцию активации ReLU. В этом случае мы настраиваем сверточную нейронную сеть для обработки входного тензора размера (28, 28, 1), который является размером изображений MNIST (третий параметр - это цветной канал, который в нашем случае - глубина 1) и мы указываем это с помощью значения аргумента*input_shape = (28, 28, 1)*в нашем первом слое:

```
from keras import layers
```

```
from keras import modelsmodel = models.Sequential()
model.add(layers.Conv2D(32,(5,5),activation='relu',
                input_shape=(28, 28,1)))
model.add(layers.MaxPooling2D((2, 2)))model.summary()
Layer (type)        Output Shape        Param #
conv2d_1 (Conv2D)  (None, 24, 24, 32)  832
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)  0
Total params: 832
Trainable params: 832
Non-trainable params: 0
```

Число параметров слоя conv2D соответствует весовой матрице W 5 × 5 и *б* Смещение для каждого из фильтров составляет 832 параметра (32 × (25 + 1)). Max-pooling не требует параметров, так как это математическая операция, чтобы найти максимум.

И чтобы построить «глубокую» нейронную сеть, мы можем сложить несколько слоев, как тот, что был построен в предыдущем разделе. Чтобы показать читателю, как это сделать в нашем примере, мы создадим вторую группу слоев, которая будет иметь 64 фильтра с окном 5 × 5 в сверточном слое и окном 2 × 2 в пуле. В этом случае число входных каналов примет значение 32 объектов, которые мы получили из предыдущего слоя, хотя, как мы видели ранее, указывать его не нужно, потому что Керас выводит его:

```
model = models.Sequential()
model.add(layers.Conv2D(32,(5,5),activation='relu',
                input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

Если архитектура модели показана *срезюме()*Метод, мы можем увидеть:

```
Layer (type)              Output Shape Param   #
conv2d_1 (Conv2D)         (None, 24, 24, 32)   832
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)   0
conv2d_2 (Conv2D)         (None, 8, 8, 64)     51264
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)     0
Total params: 52,096
Trainable params: 52,096
Non-trainable params: 0
```

В этом случае размер получающегося второго сверточного слоя составляет 8 × 8, поскольку теперь мы начинаем с входного пространства 12 × 12 × 32 и скользящего окна 5 × 5, учитывая, что он имеет шаг 1. Число параметров 51 264 соответствует тому факту, что второй уровень будет иметь 64 фильтра (как мы указали в аргументе), каждый из которых содержит 801 параметр (1 соответствует смещению, а матрица W - 5 × 5 для каждого из 32 записи). Это означает ((5 × 5 × 32) +1) × 64 = 51264.

Можно увидеть, что вывод*Conv2D*а также*MaxPooling2D*Слои - это тензор трехмерной формы (высота, ширина, каналы). Размеры ширины и высоты, как правило, уменьшаются, когда мы входим в скрытые слои нейронной сети. Количество ядер контролируется с помощью первого аргумента, передаваемого*Conv2D*слой (обычно размер 32 или 64).

Следующий шаг, теперь, когда у нас есть 64 фильтра 4x4, это добавить плотно связанный слой, который будет служить для подачи последнего слоя softmax:

model.add(layers.Dense(10, activation='softmax'))

В этом примере мы должны настроить тензоры на вход плотного слоя, такого как softmax, который является 1D-тензором, в то время как выход предыдущего - 3D-тензором. Вот почему мы должны сначала сгладить трехмерный тензор с одним из 1D. Наш вывод (4,4,64) должен быть сведен к вектору (1024) перед применением Softmax.

В этом случае число параметров слоя softmax составляет 10 × 1024 + 10 с выходом вектора 10:

model =
models.Sequential()model.add(layers.Conv2D(32,(5,5),activation='relu',
input_shape=(28,28,1)))

model.add(layers.MaxPooling2D((2, 2)))model.add(layers.Conv2D(64, (5, 5), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))model.add(layers.Flatten())

model.add(layers.Dense(10, activation='softmax'))

С *резюме()* Методом, мы можем увидеть эту информацию о параметрах каждого слоя и форме выходных тензоров каждого слоя:

Layer (type) Output Shape Param #

conv2d_1 (Conv2D) (None, 24, 24, 32) 832

max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32) 0

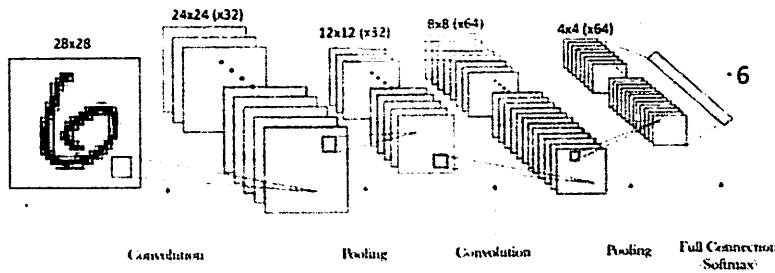
conv2d_2 (Conv2D) (None, 8, 8, 64) 51264

max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64) 0

flatten_1 (Flatten) (None, 1024) 0
dense_1 (Dense) (None, 10) 10250
Total params: 62,346
Trainable params: 62,346
Non-trainable params: 0

Наблюдая за этим кратким изложением, легко понять, что в сверточных слоях требуется больше памяти и, следовательно, больше вычислений для хранения данных. Напротив, в плотно связанном слое softmax требуется мало места в памяти, но, для сравнения, модель требует многочисленных параметров, которые необходимо изучить. Важно знать размеры данных и параметры, потому что, когда у нас есть модели, основанные на сверточных нейронных сетях, они имеют много слоев, как мы увидим позже, и эти значения могут расти экспоненциально.

Более наглядное представление вышеуказанной информации показано на следующем рисунке, где мы видим графическое представление формы тензоров, которые передаются между слоями и их соединениями:



Как только модель нейронной сети определена, мы готовы обучать модель, то есть настраивать параметры всех сверточных слоев. Отсюда, чтобы узнать, насколько хорошо работает наша модель, мы должны сделать то же самое, что и в примере с Keras в предыдущем посте «Глубокое обучение для начинающих: практическое руководство по Python и Keras». По этой причине и во избежание повторений мы будем использовать код, уже представленный выше:

from keras.datasets import mnist

from keras.utils import to_categorical(train_images, train_labels), (test_images, test_labels) =

```
mnist.load_data()train_images =
train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255test_images =
test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255train_labels =
to_categorical(train_labels)
test_labels =
to_categorical(test_labels)model.compile(loss='categorical_crossentropy',
optimizer='sgd',
metrics=['accuracy'])model.fit(train_images, train_labels,
batch_size=100,
epochs=5,
verbose=1)test_loss, test_acc = model.evaluate(test_images,
test_labels)print('Test accuracy:', test_acc)Test accuracy: 0.9704
```

Если выполнили код на компьютере только с процессором, будет замечено, что на этот раз обучение сети заняло намного больше времени, чем в предыдущем примере, даже всего с 5 эпохами. Можете ли вы представить, сколько времени может занять сеть из множества слоев, эпох или изображений? Отсюда, как мы уже обсуждали во введении к книге, нам необходимо обучить нейронные сети для реальных случаев с большим количеством вычислительных ресурсов, таких как графические процессоры.

Заметили, что в этом примере мы не отделили часть данных для проверки модели, как мы указали в предыдущем разделе, которые будут передаваться в*validation_data* аргумент *поместиться()* метод. Как это возможно?

Как мы видели в других случаях, Keras принимает много значений по умолчанию, и одним из них является это. На самом деле, если *validation_data* аргумент не указан, Keras использует *validation_split* аргумент, представляющий собой целое число от 0 до 1, указывающее долю обучающих данных, которые следует рассматривать как проверочные данные (аргумент, который мы не указали в этом примере).

Например, значение 0,2 подразумевает, что 20% данных, указанных в массивах Numpy, будут разделены в первых двух аргументах *поместиться()* метод и не будет включен в тренинг, используется только для оценки потерь или любого другого показателя в конце каждой эпохи Его значение по умолчанию составляет 0,0. То есть, если ни один из этих аргументов не указан, процесс проверки не выполняется в конце каждой эпохи.

На самом деле, нет особого смысла не делать этого, потому что гиперпараметры, которые мы передаем в качестве аргументов методам, очень важны, и именно использование данных проверки имеет решающее значение для поиска наилучшего значения. Но в этой вводной книге мы посчитали удобным не вдаваться в подробности, и вот упрощение.

Кроме того, этот пример служит для того, чтобы подчеркнуть, что Keras имеет большинство гиперпараметров, инициализированных по умолчанию, таким образом, что это облегчает начинающим внедрение нейронной сети.

Основными гиперпараметрами сверточных нейронных сетей, не замеченными до сих пор, являются: размер окна фильтра, количество фильтров, шаг и заполнение.

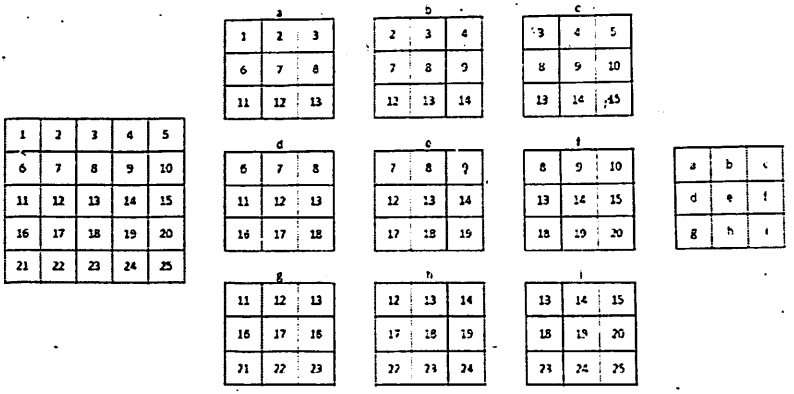
Размер окна (*window_height*×*window_width*), которая содержит информацию от пространственно близких пикселей, обычно составляет 3 × 3 или 5 × 5. Количество фильтров, которое сообщает нам количество характеристик, которые мы хотим обработать (output_depth), обычно составляет 32 или 64. В слоях Keras в Conv2D эти гиперпараметры являются теми, которые мы передаем в качестве аргументов в следующем порядке:

Conv2D (output_depth, (window_height, window_width))

Чтобы объяснить концепцию заполнения, приведем пример. Давайте предположим, что изображение размером 5 × 5 пикселей. Если мы выберем окно 3 × 3 для выполнения свертки, мы увидим, что тензор, полученный в результате операции, имеет размер 3 × 3. То есть он немного сжимается: в данном случае ровно два пикселя для каждого измерения.

На следующем рисунке это визуально отображается. Предположим, что рисунок слева - это изображение 5 × 5. В нем мы пронумеровали пиксели, чтобы было легче увидеть, как движется капля 3 × 3 для вычисления элементов фильтра. В центре показано, как окно 3 × 3 перемещалось по изображению, 2 позиции вправо и две позиции внизу.

Результат применения операции свертки возвращает фильтр, который мы представили слева. Каждый элемент этого фильтра помечается буквой, которая связывает его с содержимым скользящего окна, для которого рассчитывается его значение.

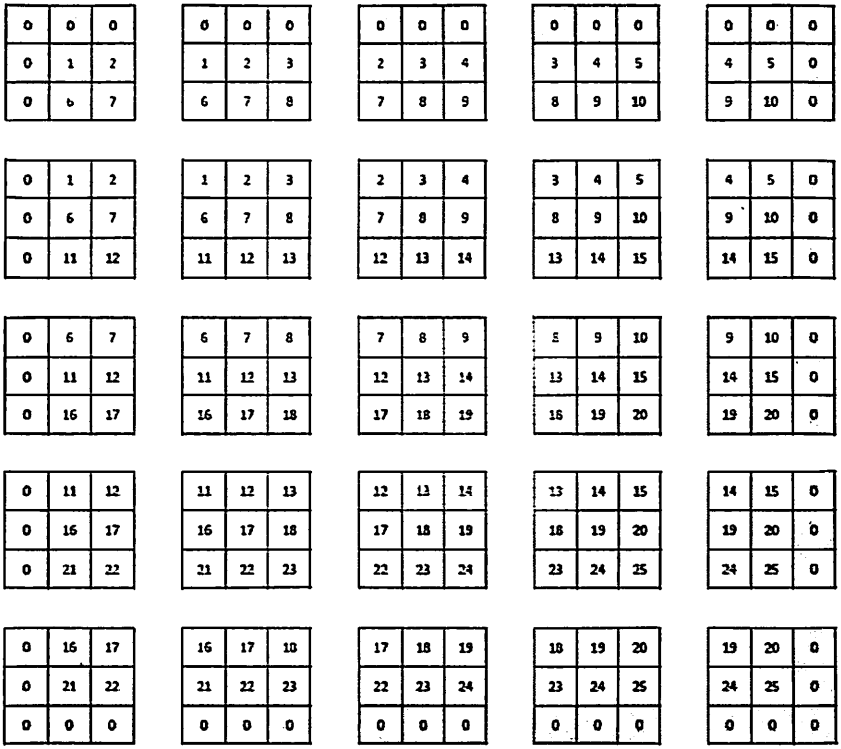


Этот же эффект можно наблюдать на примере сверточной нейронной сети, который мы создаем в этом посте. Мы начинаем с входного изображения 28 × 28 пикселей, а результирующие фильтры - 24 × 24 после первого слоя свертки. И во втором слое свертки мы прошли путь от натяжителя 12 × 12 до натяжителя 8 × 8.

Но иногда мы хотим получить выходное изображение с теми же размерами, что и входные, и для этого мы можем использовать заполнение гиперпараметра в сверточных слоях. С отступом мы можем добавить нули вокруг входных изображений, прежде чем скользить через них через окно. В нашем случае на предыдущем рисунке, чтобы выходной фильтр имел тот же размер, что и входное изображение, мы можем добавить столбец справа, столбец слева, строку выше и строку ниже к входному изображению нулей. , Визуально это видно на следующем рисунке:

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	16	17	18	19	20	0
0	21	22	23	24	25	0
0	0	0	0	0	0	0

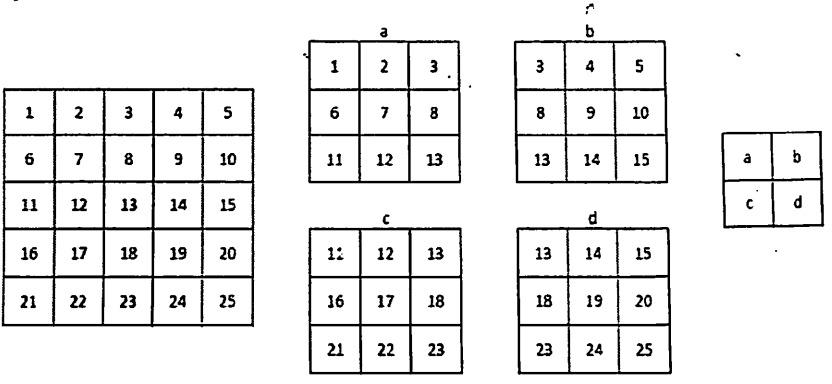
Если мы теперь сдвинем окно 3 × 3, мы увидим, что оно может сдвинуть 4 позиции вправо и 4 позиции вниз, генерируя 25 окон, которые генерируют размер фильтра 5 × 5.



В Keras заполнение в слое Conv2D настроено с аргументом заполнения, который может иметь два значения: «то же самое», которое указывает, что столько строк и столбцов нулей добавляется по мере необходимости, чтобы выходные данные имели то же измерение, что и запись; и «допустимый», который указывает на отсутствие заполнения (который является значением по умолчанию этого аргумента в Keras).

Другим гиперпараметром, который мы можем указать в сверточном слое, является шаг, который указывает количество шагов, в которые перемещается окно фильтра (в предыдущем примере шаг был один).

Большие значения шага уменьшают размер информации, которая будет передана следующему слою. На следующем рисунке мы можем видеть тот же предыдущий пример, но теперь со значением шага 2:



Как мы видим, изображение 5 × 5 стало меньшим фильтром 2 × 2. Но в действительности сверточные шаги по уменьшению размеров редко используются на практике; для этого используются операции объединения, которые мы представили ранее. В Keras шаг в слое Conv2D настроен с аргументом шага, который по умолчанию равен *шагает = (1,1)* значение, которое отдельно указывает на прогресс в двух измерениях.

Лабораторная работа №5. Использование инструмента визуализации TensorBoard

TensorBoard - это структура визуализации тензорного потока для понимания и проверки потока алгоритма машинного обучения.

Оценка модели машинного обучения может быть выполнена по многим показателям, таким как потери, точность, график модели и многое другое. Производительность алгоритма машинного обучения зависит от выбора модели и гиперпараметров, введенных в алгоритм. Эксперименты выполняются путем изменения значений этих параметров.

Модели глубокого обучения подобны черному ящику, в нем трудно найти обработку. Важно понять, как построить модель. С помощью визуализации вы можете узнать, какие параметры нужно изменить, на какую величину получить улучшение производительности модели. Таким образом, TensorBoard является важным инструментом для визуализации каждой эпохи на этапе обучения модели.

Чтобы установить тензорную доску с помощью pip, выполните следующую команду:

pip install tensorboard

Кроме того, он может быть установлен с помощью команды conda,

Conda install tensorboard

Использование тензорной доски с моделью Keras:

Keras - это библиотека с открытым исходным кодом для моделей глубокого обучения. Это высокоуровневая библиотека, которую можно запустить поверх tenorflow, theano и т. Д.

Чтобы установить tenorflow и библиотеку Keras с помощью pip:

pip install tensorflow pip install Keras

Давайте рассмотрим простой пример классификации с использованием набора данных MNIST. MNIST - это английский набор цифровых данных, содержащий изображения чисел от 0 до 9. Он доступен с библиотекой Keras.

- Импортируйте тензор потока библиотеки, так как мы будем использовать Keras с бэкэндом тензор потока.

import tensorflow as tf

- Сначала загрузите набор данных MNIST из Keras в набор обучающих и тестовых данных.

mnist = tf.keras.datasets.mnist

- Последовательная модель создается с использованием, tf.keras.models.Sequential

- Для обучения используется модель Model.fit (). Журналы могут быть созданы и сохранены с использованием.

tf.keras.callback.TensorBoard

- Чтобы включить вычисление гистограммы,

histogram_freq=1.

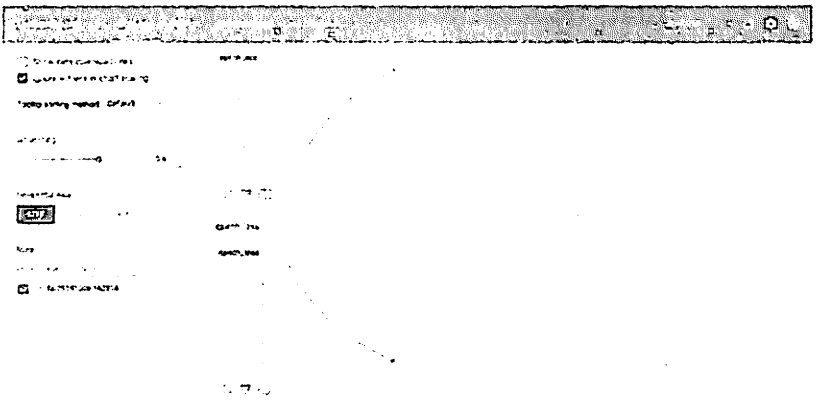
По умолчанию он выключен.

Код для вышеупомянутой классификации набора данных MNIST выглядит следующим образом:

```
# Simple NN to classify handwritten digits from MNIST dataset
import tensorflow as tf
import datetime
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data() x_train, x_test = x_train / 255.0, x_test / 255.0
def create_model(): return tf.keras.models.Sequential(( tf.keras.layers.Flatten(input_shape=(28, 28)), tf.keras.layers.Dense(512, activation='relu'), tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10, activation='softmax')
))
model = create_model() model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=('accuracy'))
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
model.fit(x=x_train, y=y_train, epochs=5, validation_data=(x_test, y_test),
callbacks=(tensorboard_callback))
```

Чтобы запустить тензорную доску на локальном сервере, перейдите в каталог, где установлен тензорный поток, а затем выполните следующую команду:

```
tensorboard --logdir=/path/to/logs/files
```



Скаляры показывают изменения с каждой эпохой. На приведенном выше рисунке показан график точности и потерь после каждой эпохи. Epoch_acc и epoch_loss - это точность обучения и потеря обучения. Тогда как epoch_val_acc и epoch_val_loss - это точность и потеря данных проверки.

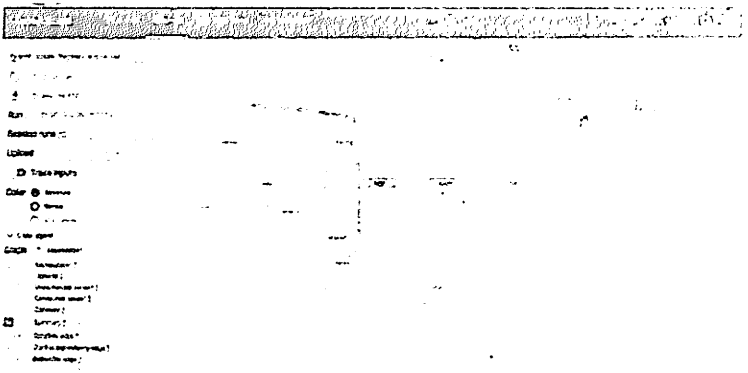
Более светлые оранжевые линии показывают точную точность или потерю, а более темные - сглаженные значения. Сглаживание помогает визуализировать общую тенденцию в данных.

Страница График помогает вам визуализировать график вашей модели. Это поможет вам проверить, правильно ли построена модель или нет.

Чтобы визуализировать график, нам нужно создать сеанс, а затем объект TensorFLow FileWriter. Чтобы создать объект записи, нам нужно передать путь, где хранится сводка, и sess.graph в качестве аргумента.

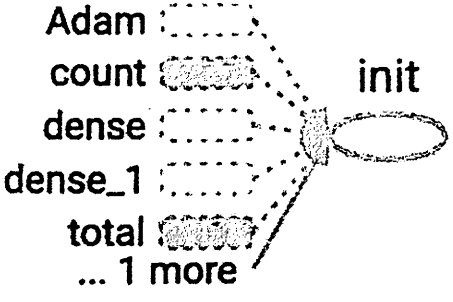
writer = tf.summary.FileWriter(STORE_PATH, sess.graph)

tf.placeholder () и tf.Variable () используются для заполнителей и переменных в коде тензорного потока.

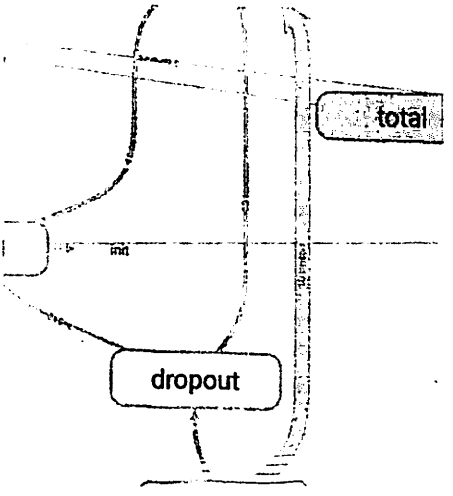


Это показывает графическую визуализацию модели, которую мы построили. Все прямоугольники со скругленными углами являются пространствами имен. А овалы показывают математические операции.

Константы показаны в виде маленьких кружков. Чтобы уменьшить беспорядок в графике, тензорная доска делает некоторые упрощения, используя пунктирные овалы или скругленные прямоугольники с пунктирными линиями. Это узлы, которые связаны со многими другими узлами или со всеми узлами. Таким образом, они хранятся в виде точек на графике, а их детали можно увидеть в правом верхнем углу. В верхнем правом углу указана связь с градиентами, градиентными спусками или узлами инициализации.



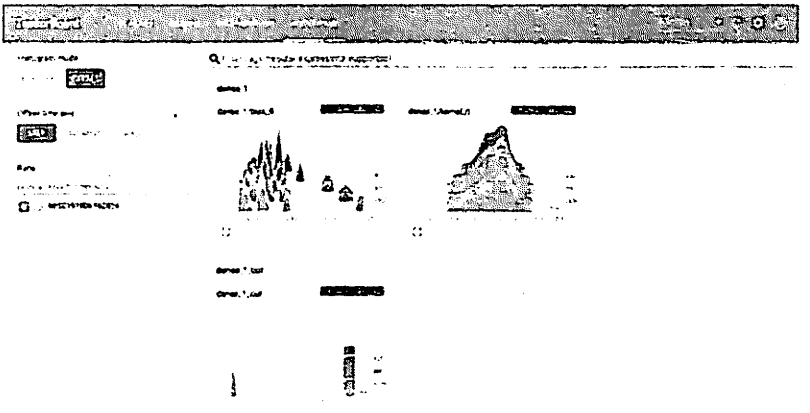
Чтобы узнать количество тензоров, входящих и выходящих из каждого узла, вы можете увидеть ребра на графике. Ребра графа описывают количество тензоров, протекающих в графе. Это помогает идентифицировать входные и выходные размеры каждого узла. Это помогает в отладке любой проблемы.



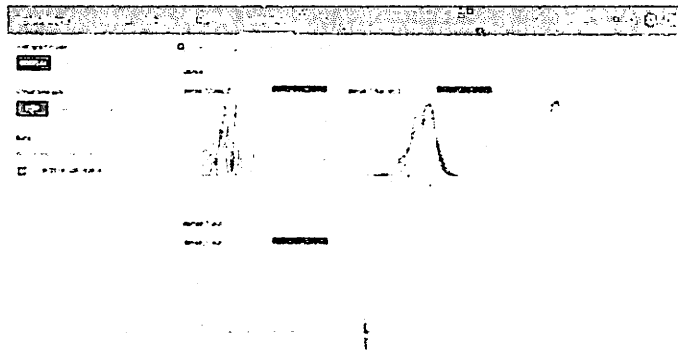
Это показывает тензорные распределения во времени, а также мы можем видеть веса и смещения. Это показывает прогресс входов и выходов с течением времени для каждой эпохи. Есть два варианта просмотра:

Смещение и наложение.

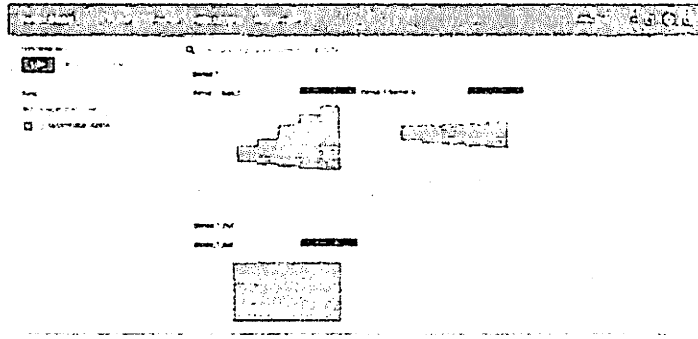
Смещение гистограммы будет следующим:



Вид наложения гистограммы:



Страница Распределение показывает статистические распределения. График показывает среднее значение и стандартные отклонения.



- TensorBoard помогает визуализировать процесс обучения, записывая резюме модели, такие как скаляры, гистограммы или изображения. Это, в свою очередь, помогает повысить точность модели и легко отлаживать.
- Обработка глубокого обучения - это черный ящик, а тензорная доска помогает понять обработку, происходящую в черном ящике, с помощью графиков и гистограмм.

TensorBoards предоставляет визуализацию для модели глубокого заработка, которая обучается и помогает в их понимании. Может использоваться как с TensorFlow, так и с Keras. Он в основном обеспечивает визуализацию поведения скаляров, метрик с помощью гистограмм и графа модели в целом.

1. Информационные технологии и вычислительные системы: Обработка информации и анализ данных. Программная инженерия. Математическое моделирование. Прикладные аспекты информатики / Под ред. С.В. Емельянова. - М.: Ленанд, 2015. - 104 с.

2. Труды ИСА РАН: Математические модели социально-экономических процессов. Динамические системы. Управление рисками и безопасностью. Оптимизация, идентификация, теория игр. Обработка и анализ изображений и сигналов. Интеллектуальный анализ данных и распознав / Под ред. С.В. Емельянова. - М.: Красанд, 2013. - 128 с.

3. Агапов, А.В. Обработка и обеспечение безопасности электронных данных: Учебное пособие / А.В. Агапов, Т.В. Алексеева и др. - М.: МФПУ Синергия, 2012. - 592 с.

4. Агаянц, И.М. Азы статистики в мире химии: Обработка экспериментальных данных / И.М. Агаянц. - СПб.: Профессия, 2015. - 614 с.

5. Бурнаева, Э., Г. Обработка и представление данных в MS Excel: Учебное пособие / Э.Г. Бурнаева, С.Н. Леора. - СПб.: Лань, 2016. - 160 с.

6. Волкова, П.А. Статистическая обработка данных в учебно-исследовательских работах / П.А. Волкова, А.Б. Шипунов. - М.: Форум, 2012. - 96 с.

7. Волкова, П.А. Статистическая обработка данных в учебно-исследовательских работах: Учебное пособие / П.А. Волкова, А.Б. Шипунов. - М.: Форум, 2017. - 832 с.

8. Громов, Ю.И. Обработка и представление данных в MS Excel: Учебное пособие / Ю.И. Громов. - СПб.: Лань, 2016. - 160 с.

9. Исакова, О.П. Обработка и визуализация данных физических экспериментов с помощью пакета Origin / О.П. Исакова, Ю.Ю. Тарасевич, Ю.И. Юзюк. - М.: КД Либроком, 2019. - 136 с.

10. Исакова, О.П. Обработка и визуализация данных физических экспериментов с помощью пакета Origin / О.П. Исакова, Ю.Ю. Тарасевич, Ю.И. Юзюк. - М.: КД Либроком, 2009. - 136 с.

11. Крянев, А.В. Метрический анализ и обработка данных / А.В. Крянев, Г.В. Лукин, Д.К. Удумян. - М.: Физматлит, 2012. - 308 с.

12. Лацис, А.О. Параллельная обработка данных: Учебное пособие для студентов вузов / А.О. Лацис. - М.: ИЦ Академия, 2010. - 336 с.

13. Лацис, А.О. Параллельная обработка данных / А.О. Лацис. - М.: Academia, 2017. - 456 с.

14. Нархид, Н. Apache Kafka. Потоковая обработка и анализ данных / Н. Нархид. - СПб.: Питер. 2019. - 320 с.

15. Сенько, А Работа с BigData в облаках. Обработка и хранение данных с примерами из MIcrosoft Azure / А Сенько. - СПб.: Питер. 2019. - 448 с.

16. Сенько, А. Работа с BIGDATA в облаках.Обработка и хранение данных с примерами из Microsoft / А. Сенько. - СПб.: Питер, 2019. - 448 с.

17. Сенько, А.В. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure / А.В. Сенько. - СПб.: Питер, 2018. - 126 с.

18. Уикем, Х. Язык R в задачах науки о данных: импорт, подготовка, обработка, визуализация и моделирование данных / Х. Уикем. - М.: Диалектика, 2018. - 592 с.

19. Чашкин, Ю.Р. Математическая статистика. Анализ и обработка данных: Учебное пособие / Ю.Р. Чашкин; Под ред. С.Н. Смоленский. - Рн/Д: Феникс, 2010. - 236 с.

Оглавление

Введение 3

Лабораторная работа №1. Изучение программного языка Python. 4

Лабораторная работа №2. Формирование таблицы об распределении данных 13

Лабораторная работа №3. Изучение возможностей библиотек TensorFlow в Python 29

Лабораторная работа №4. Визуализация процессов обучения нейронных сетей и принятия ими решений 50

Лабораторная работа №5. Использование инструмента визуализации TensorBoard 77

ЛИТЕРАТУРА 83

Д.К.МУХАМЕДИЕВА, Н.А.НИЁЗМАТОВА, М.Ю.ДОШАНОВА

Учебно-методическое пособие по дисциплине

«ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ»

Редактор – И. Халилов

Компьютерная верстка – М. Хакимов

Издательство «Fan ziyosi»
Лицензия № 308197041 от 14 февраля 2021 года.
Адрес изд: г.Ташкент, ул.А.Навоий-30.

Бумага офсет. Формат 60x84.$^{1/16}$ Усл. печ.л. 5.5.
Гарнитура Times New Roman. Офсетная печать.
Заказ № 57. 08.11.2022 год. Тираж 300.
Отпечатано в типографии ООО «Munis design group»
100170, г.Ташкент, ул. Буз-2, проезд, дом-17-А.