

# Машинное обучение (Machine Learning)

## Глубокое обучение: Сверточные сети (Deep Learning: Convolutional nets)

Уткин Л.В.

Санкт-Петербургский политехнический университет Петра Великого



# Содержание

- 1 Причины создания сверточных сетей и предварительные замечания
- 2 Сверточный слой
- 3 Max-pooling
- 4 Некоторые особенности и детали сверточных сетей



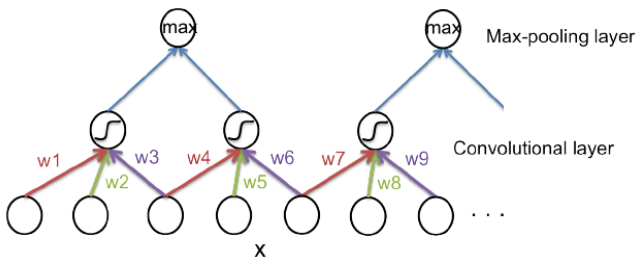






# Сверточные сети

К свертке добавляется другой слой, известный как **“max-pooling layer”**, который вычисляет максимальное значение из выбранного подмножества выходных нейронов сверточного слоя и использует его как входное значение следующего слоя



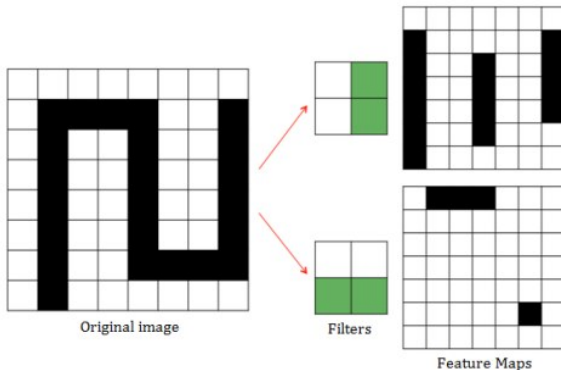








## Сверточный слой (фильтр)

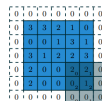
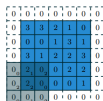
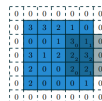
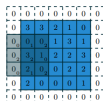
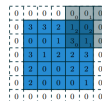
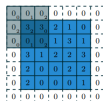


Цель - определение горизонтальных и вертикальных границ.  
Используем фильтры (зеленый цвет) – это небольшая матрица, представляющая признак, который хотим найти на исходном изображении.





# Padding




















## Сверточный слой (фильтр)

Operation	Filter	Convolved Image
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



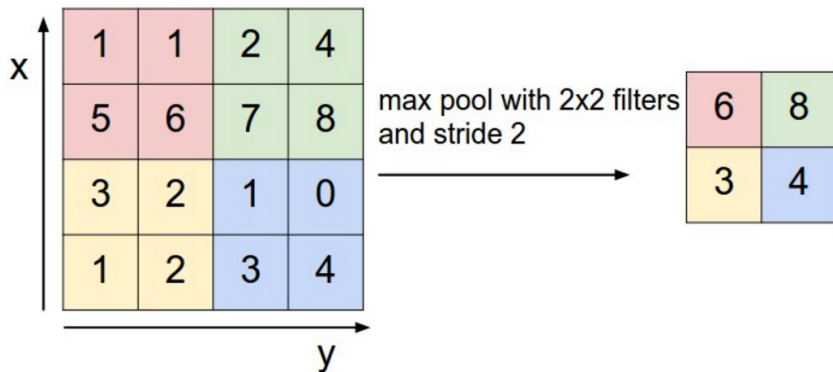
# Max-pooling

- По мере продвижения вглубь сверточной нейронной сети, уточняем информацию на картах признаков. Если признак присутствует на предыдущем слое, полученная карта содержит ключевые пиксели, окруженные нечетким «ореолом», в рамках которого признак может быть определен не до конца.
- Решение - метод - max-объединение (max-pooling). Это - разделение карты признаков на непересекающиеся участки и выделения на этих участках нейронов с максимальной активностью. Max-pooling карты признаков делает распознавание более точным, избавляясь от ненужных «ореолов».





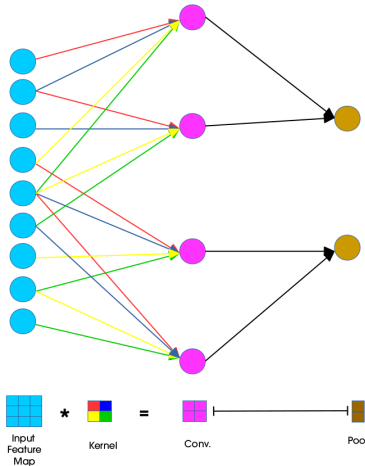
# Max-pooling



## Субдискретизирующий слой

- Выполняет уменьшение размера входной карты признаков, например в 2 раза.
- Разные методы, например, метод выбора максимального элемента (max-pooling) - вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению.
- Формально  $\mathbf{x}^{(k)} = f(W^{(k)} \cdot \text{subsample}(\mathbf{x}^{(k-1)}) + b^{(k)})$ ,  $f$  - функция активации,  $\text{subsample}$  - операция выборки локальных максимальных значений.
- Использование этого слоя позволяет улучшить распознавание примеров с измененным масштабом (уменьшенных или увеличенных).

## Представление свертки и пулинга





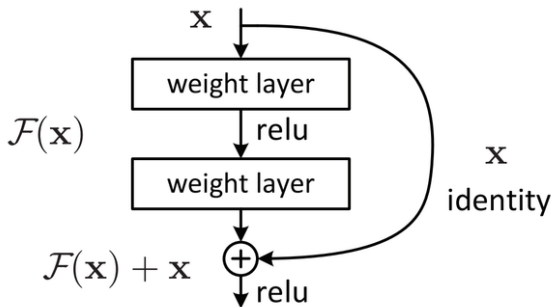




## Residual block

- Сеть обучается предсказывать функцию  $F(x) - x$ , вместо функции  $F(x)$ , которую изначально нужно было предсказывать
- Для компенсации этой разницы вводится это замыкающее соединение (shortcut connection), которое добавляет недостающий  $x$  к функции.
- Такая разностная функция будет проще обучать, чем исходная.

## Residual block (ResNet)





# Рецептивные поля (1)

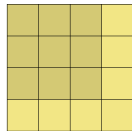
- Рецептивное поле — часть матрицы входных сигналов, подвергаемая “свертыванию”.
- Особенность сверточной сети - размеры ввода становятся все меньше и меньше от начала до конца сети, а количество каналов становится больше
- Receptive field определяет, какую область исходного входа получает выход
- Идея strided convolution - обрабатываем пролеты только на фиксированном расстоянии друг от друга и пропускаем те что посередине
- Оставляем только выходы на определенном расстоянии друг от друга, удаляя остальные

## Рецептивные поля (2)

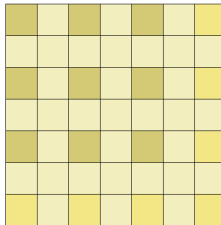
- Процесс свертывания - уплотнит количества данных, заложенных в один пиксел
- Пусть имеется матрица импульсов размерностью  $3 \times 3$  (9 единиц информации)
- Использование свертки с ядром  $2 \times 2$  “свернет” входные импульсы в матрицу  $2 \times 2$ , что даст 4 единицы информации
- Информация в девяти ячейках теперь помещается в меньшем пространстве, в четырех ячейках
- Потери

## Рецептивные поля (3)

- Применяем нелинейность к выходным данным, накладываем еще один новый слой свертки сверху.
- Даже если бы применили ядро того же размера ( $3 \times 3$ ), имеющее одну и ту же локальную область, к выходу strided convolution, ядро имело бы более эффективное receptive field



Operation on the true strided  
output



Operation on the output with the removed pixels

## Рецептивные поля

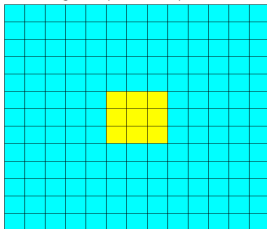
- Это связано с тем, что выход strided слоя по-прежнему представляет из себя одно и то же изображение.
- Это не столько обрезка, сколько изменение размера, только теперь каждый отдельный пиксель на выходе является «представителем» большей площади (другие пиксели которой были отброшены) из того же местоположения исходного ввода.
- Поэтому, когда ядро следующего слоя применяется к выходу, оно работает с пикселями, собранными из большей области.
- Такое расширение поля позволяет слоям свертки сочетать признаки низкого уровня (линии, ребра) с признаками более высокого уровня (кривые, текстуры)

## Расширяемая свертка (Dilated convolution)

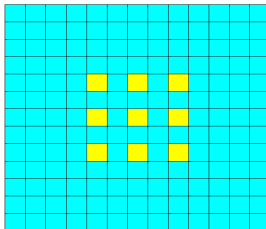
- Расширяемая свертка решает задачу посредством дополнительного слоя нейронной сети
- Похожа на пуллинг и свертку с шагом, но позволяет:
  - расширить рецептивное поле без потери качества изображения
  - получить большее рецептивное поле при тех же затратах на вычисления и расходах памяти, при этом сохранив качество изображения
- Формула свертки:  $O_{x,y} = \sum_{i,j} w_{ij} \sum_{|t-i| \leq l, |k-j| \leq l}^{t,k} I_{t,k}$
- $I$  - входные данные,  $O$  - выходные,  $W$  - ядро свертки,  $l$  - коэффициент расширения.

# Dilated convolution

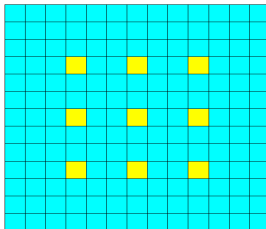
D=1, regular (standard) convol



D=2

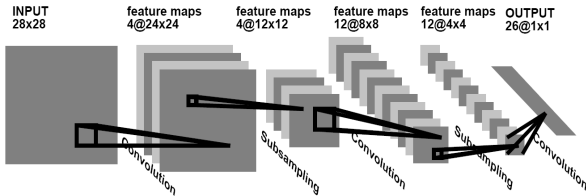


D=3



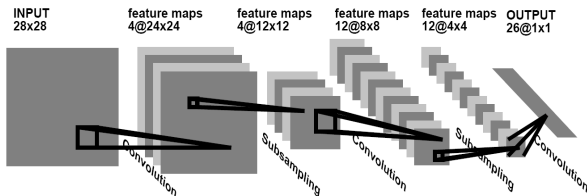
1-, 2- и 4-расширенные свертки с классическими ядрами 3x3, 5x5 и 9x9. Желтые точки обозначают ненулевые веса, остальные веса ядра равны нулю. Выделенные области обозначают рецептивные поля.

## Сверточная сеть в целом (1)



- 1 входной слой - матрица картинки  $28 \times 28$
- 2 сверточный слой - набор одностипных матриц (карт признаков) по числу ядер свертки
- 3 субдискретизирующий слой - уменьшенный в 2 раза предыдущий набор матриц

## Сверточная сеть в целом (2)



- 4 сверточный слой - предыдущий набор матриц суммируется в соответствии со схемой соединения слоев и генерируется новый набор по числу ядер свертки
- 5 субдискретизирующий слой - уменьшенный в 2 раза предыдущий набор матриц
- 6 выходной слой - предыдущий набор матриц разворачивается в вектор и обрабатывается





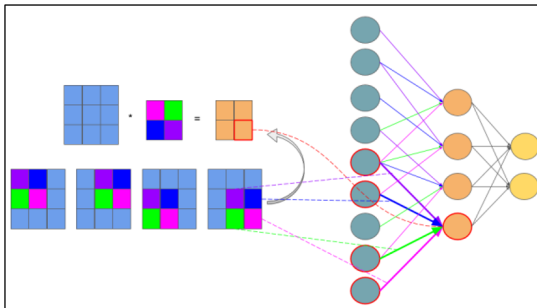
## Пример (программа)

```
def generate_cnn_model():
    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),
        Conv(fshape=(3, 3, 32), padding=2, strides=1,
            dilation=2, init=Kaiming(), activation=Rectlin()),
        Pooling(fshape=(3, 3), padding=1, strides=2, op='max'),
        Conv(fshape=(3, 3, 64), padding=2, strides=1,
            dilation=2, init=Kaiming(), activation=Rectlin()),
        Pooling(fshape=(3, 3), padding=1, strides=2, op='max'),
        Affine(nout=class_count, init=Xavier(),
            activation=Logistic(shortcut=True)) ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=CrossEntropyBinary())
    return (model, cost)
```

# Обучение сети

1. Случай: сверточный слой находится перед полносвязным, тогда он имеет нейроны и связи такого же типа, как в полносвязном слое, соответственно вычисление  $\delta$  ошибки ничем не отличается от вычисления  $\delta$  скрытого слоя.
2. Случай: сверточный слой находится перед сверточным, вычисление  $\delta$  происходит путем обратной свертки.

# Обучение сети - случай 1



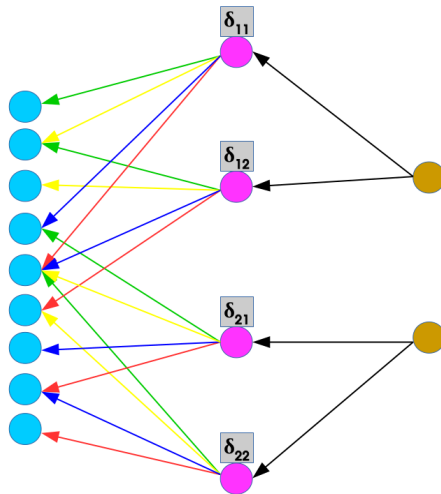
Интерпретация операции свертки в многослойный вид, где связи с одинаковым цветом имеют один и тот же вес. Синим цветом обозначена подвыборочная карта, разноцветным – синаптическое ядро, оранжевым – получившаяся свертка

## Обучение сети - случай 2

**Обратная свертка** – это тот же способ вычисления дельт, но различие в повороте ядра на 180 градусов и скользящем процессе сканирования сверточной карты дельт с измененными краевыми эффектами, т.е. необходимо взять ядро сверточной карты (следующего за подвыборочным слоем) повернуть его на 180 градусов и сделать обычную свертку по вычисленным ранее дельтам сверточной карты, но так чтобы окно сканирования выходило за пределы карты.



# Обучение сети - случай 2



Input

\*



Gradients

=



Kernel



Pool

# Особенность соединения слоев и топология сетей (1)

- Сверточную сеть из 7 слоев:
  - 1 входной слой - матрица картинки
  - 2 сверточный слой - набор однотипных матриц (карт признаков)
  - 3 субдискретизирующий слой - уменьшенный в 2 раза предыдущий набор матриц
  - 4 сверточный слой - предыдущий набор матриц объединяется в одну, в соответствии со схемой соединения слоев, и генерируется новый набор
  - 5 субдискретизирующий слой - уменьшенный в 2 раза предыдущий набор матриц
  - 6 слой MLP - предыдущий набор матриц разворачивается в вектор и обрабатывается как MLP (многослойный перцептрон)
  - 7 слой MLP (выходной)



## Особенность соединения слоев (2)

При этом, нейроны (карты признаков) второго субдискретизирующего слоя и третьего сверточного слоя соединяются выборочно в соответствии с матрицей смежности, которая задается как параметр сети. Для сети с количеством карт признаков во втором слое 7 и 9 в третьем слое:

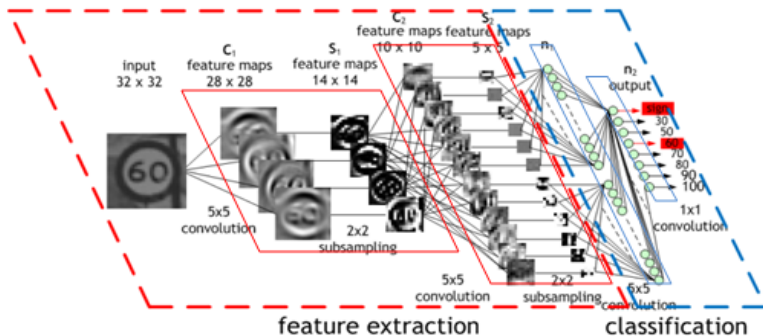
	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	1	1	1	0
2	1	1	0	0	0	0	1	1	1
3	1	1	1	0	0	0	0	1	1
4	0	1	1	1	0	0	0	1	1
5	0	0	1	1	1	0	0	0	1
6	0	0	0	1	1	1	0	0	0
7	0	0	0	0	1	1	1	0	0

## Особенность соединения слоев (3)

- Каждая выходная карта формируется частичной суммой результатов сверток входных карт, для каждой такой частичной суммы свой набор ядер свертки.
- Соединение всех карт второго слоя со всеми картами третьего слоя значительно увеличило бы количество связей.
- Соединение карт одна к одной стало бы еще одним повторением свертки, которое уже присутствовало между слоями.

# Сверточная сеть

- Фильтрация 4-мя 5x5 сверточными ядрами, создающими 4 карты признаков.
- Max-pooling
- Фильтрация 10-ю 5x5 сверточными ядрами, max-pooling



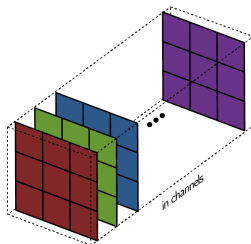
# Сверточные сети (формально)

Дано: большие изображения  $x_{\text{large}}$  размера  $r \times c$

- 1 обучаем прореженный автокодер на малых “кусочках”  $x_{\text{small}}$  размера  $a \times b$
- 2 обучаем  $k$  признаков  $f = \sigma(W^{(1)}x_{\text{small}} + b^{(1)})$  ( $\sigma$  - сигмоид), при наличии весов  $W^{(1)}$  и  $b^{(1)}$  из входа в нейроны скрытого слоя
- 3 для каждого “кусочка”  $x_s$  размера  $a \times b$  в большом изображении вычисляем  $f_s = \sigma(W^{(1)}x_s + b^{(1)})$
- 4 откуда получаем  $f_{\text{convolved}}$  - матрицу “свернутых” признаков размера  $k \times (r - a + 1) \times (c - b + 1)$

# Многоканальная версия сверточной нейронной сети

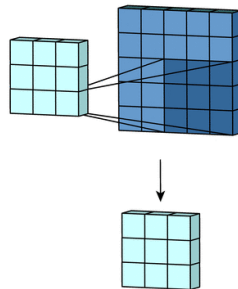
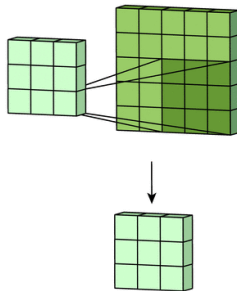
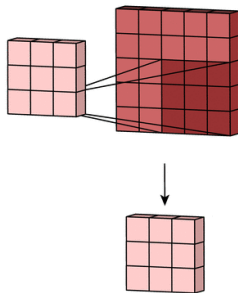
- На практике большинство входных изображений имеют 3 канала (RGB)
- Каждый фильтр - коллекция ядер, причем для каждого отдельного входного канала этого слоя есть одно ядро, и каждое ядро уникально.



# Многоканальная версия сверточной нейронной сети

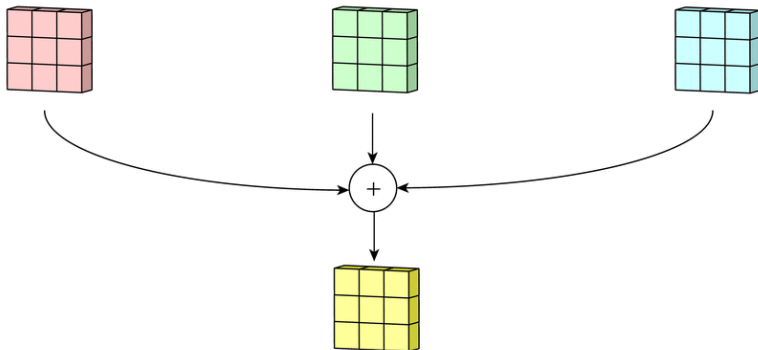
- Каждый фильтр в сверточном слое создает только один выходной канал: каждое из ядер фильтра «скользит» по их соответствующим входным каналам
- Некоторые ядра могут иметь больший вес, чем другие, для того чтобы уделять больше внимания определенным входным каналам (например, фильтр может задать красному каналу ядра больший вес, чем другим каналам).

# Многоканальная версия сверточной нейронной сети



# Многоканальная версия сверточной нейронной сети

Каждая из обработанных в канале версий суммируется вместе для формирования одного канала.





## Преимущества сверточных сетей

- Один из лучших алгоритмов по распознаванию и классификации изображений.
- По сравнению с обычной нейронной сетью гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы делать для каждого пикселя входного изображения свои весовые коэффициенты.
- Нейросеть при обучении обобщает информацию, а не попиксельно запоминает каждую картинку в весовых коэффициентах, как перцептрон.

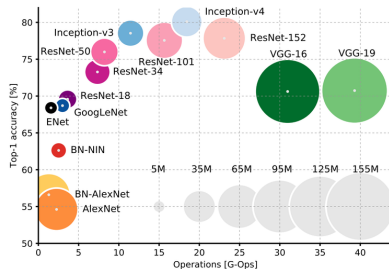
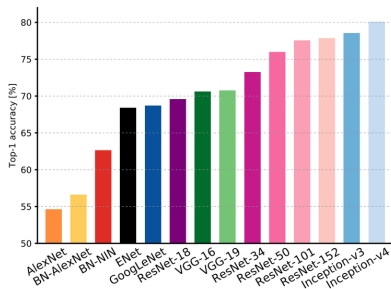
## Преимущества сверточных сетей

- Удобное распараллеливание вычислений, возможность реализации на графических процессорах.
- Относительная устойчивость к повороту и сдвигу распознаваемого изображения.
- Обучение при помощи классического метода обратного распространения ошибки.

## Недостатки сверточных сетей

- 1 Архитектура сверточной сети по большей части для распознавания изображений.
- 2 Слишком много варьируемых параметров сети: количество слоев, размерность ядра свертки для каждого из слоев, количество ядер для каждого из слоев, шаг сдвига ядра при обработке слоя, необходимость слоев субдискретизации, степень уменьшения ими размерности, функция по уменьшению размерности (выбор максимума или среднего) и т.д. Выбираются эмпирически.

# Типовые сверточные сети



<http://www.topbots.com/14-design-patterns-improve-convolutional-neural-network-cnn-architecture>

# Программное обеспечение Deep Learning: Torch

- Torch основан на библиотеке Lua
- Обработка естественного языка с помощью глубоких нейронных сетей
- Используется в Facebook и Twitter Research для исследований и разработки систем глубокого обучения

# Программное обеспечение Deep Learning: MxNet

- MxNet - мощная библиотека, поддерживающая различные языки программирования: Python, Scala, R
- Одна из самых эффективных по быстродействию и по использованию памяти библиотек
- Простота использования нескольких графических процессоров (GPU)

# Theano

- Theano - Python-библиотека
- Объединяет Keras и Lasagne
- Охватывает не только глубокое обучение, но и различные методы машинного обучения: рекуррентная нейронная сеть, ограниченная машина Больцмана, глубокие сети доверия, сверточные нейронные сети
- Проста для разработчиков
- Имеются скрипты для конвертации моделей Caffe

# Программное обеспечение Deep Learning: Lasagne

- Lasagne - библиотека для построения и обучения нейронных сетей в Theano
- Проста в использовании, понимании и расширении
- Для установки требует сначала установить Python и Theano



# Программное обеспечение Deep Learning: Keras

- Keras - модульная библиотека для построения нейронных сетей для Python
- Запускается “поверх” либо TensorFlow, либо Theano

# Программное обеспечение Deep Learning: Caffe

- Caffe - флагман глубокого обучения
- Первая успешная открытая реализация с мощной, но простой базой: нет необходимости знать код для использования Caffe, используются простые файлы описаний сети
- Не поддерживает GPU, отличные от Nvidia

# Программное обеспечение Deep Learning: TensorFlow

- TensorFlow - открытая библиотека для анализа представления данных в виде графа
- Вершины графа - математические операции, крайние вершины - матрицы данных большой размерности (тензоры)
- TensorFlow разработана в Google Brain Team в целях проведения исследования в области машинного обучения и глубоких нейронных сетей

# Программное обеспечение Deep Learning: Deeplearning4j

- Deeplearning4j (DL4j) - JVM-фреймворк (Java Virtual Machine) для решения задач, связанных с большими данными

