Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Образовательный курс
«Введение в глубокое обучение с использованием Intel® neon™ Framework»

Сверточные нейронные сети. Глубокие остаточные сети

При поддержке компании Intel

Кустикова Валентина,
к.т.н., ст.преп. каф. МОСТ ИИТММ,
ННГУ им. Н.И. Лобачевского

Содержание

- Операция «свертки»
- Общая структура сверточного слоя
- Входные и выходные данные сверточной сети
- Метод обратного распространения ошибки для сверточных нейронных сетей
- Определение количества обучаемых параметров. Оценка объема памяти, необходимого для хранения сети
- Пример применения сверточных нейронных сетей для решения задачи предсказания пола человека по фотографии
- Принципы построения сверточных сетей
- Проблема деградация модели. Глубокие остаточные сети. Пример простейшей сети

ОПЕРАЦИЯ «СВЕРТКИ». ОБЩАЯ СТРУКТУРА СВЕРТОЧНОГО СЛОЯ

Сверточные нейронные сети

- ***Сверточные нейронные сети*** – вид нейронных сетей, которые хотя бы на одном из своих слоев в качестве преобразования используют операцию «свертки»

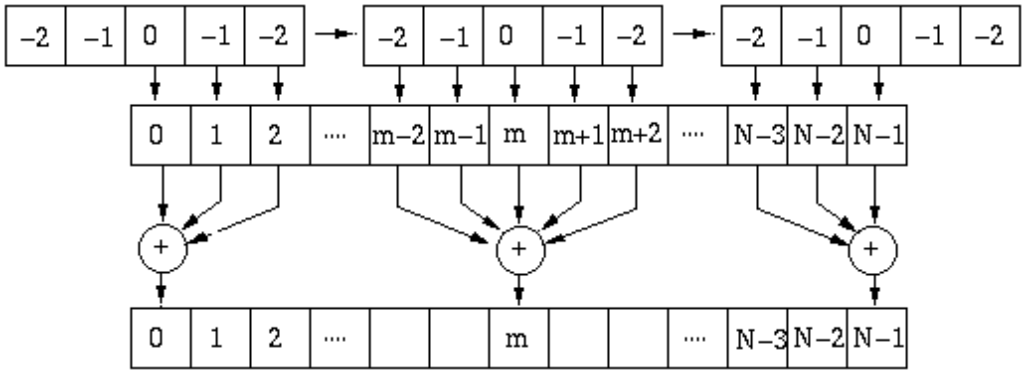
- ***Свертка*** – операция, применяемая к двум вещественнозначным функциям

Операция «свертки» (1)

- Данные в компьютерах дискретные, и измерения выполняются с некоторым интервалом, поэтому, как правило, рассматривается ***дискретная свертка***:

$$s(t) = \langle x * w \rangle(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)\,,$$

где $x(\cdot)$ – ***вход***, функция $w(\cdot)$ – ***ядро свертки***, выход – ***карта признаков*** (feature map)

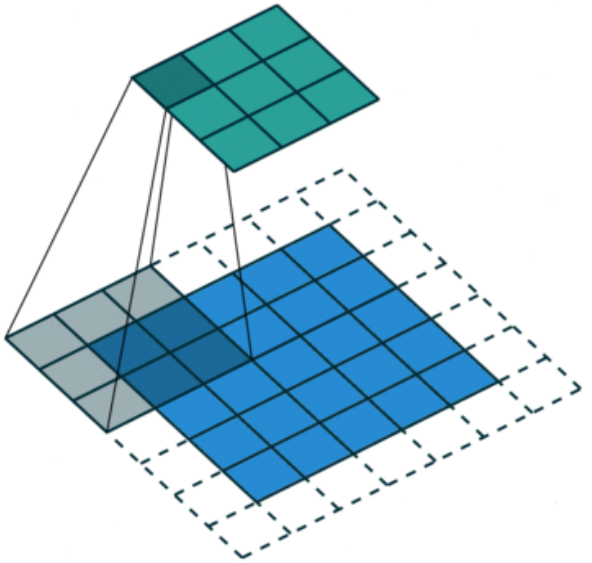


* Digital Convolution -- E186 Handout [http://fourier.eng.hmc.edu/e161/lectures/convolution/index.html].

Операция «свертки» (2)

- В задачах машинного обучения вход – многомерный массив данных (тензор), а ***ядро*** – многомерный массив параметров
- Если на входе имеется двумерное изображение I и ядро K, то операция свертки выглядит следующим образом:

$$s(i,j) = \langle I * K \rangle (i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

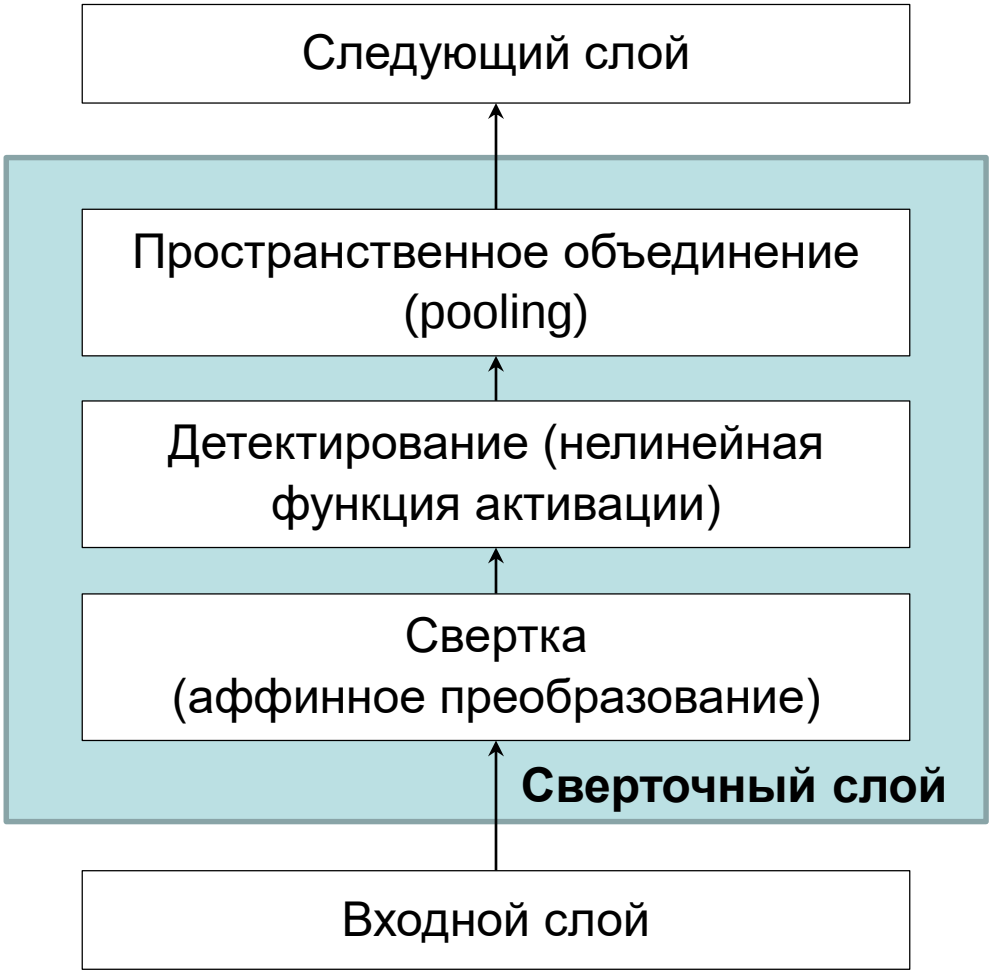


* A technical report on convolution arithmetic in the context of deep learning [https://github.com/vdumoulin/conv_arithmetic].

Общая структура сверточного слоя (1)

- Слой сверточной нейронной сети состоит из трех стадий:
  - ***Создание набора линейных активаций*** посредством выполнения одной или нескольких параллельных сверток
  - ***Детектирование*** – применение нелинейной функции активации ко всем линейным активациям
  - ***Пространственное объединение*** (pooling) с целью модификации выхода для передачи на следующий слой сети

Общая структура сверточного слоя (2)

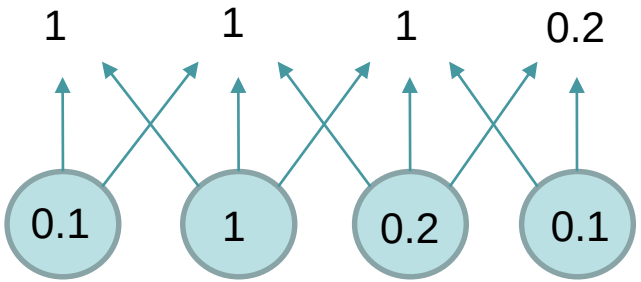


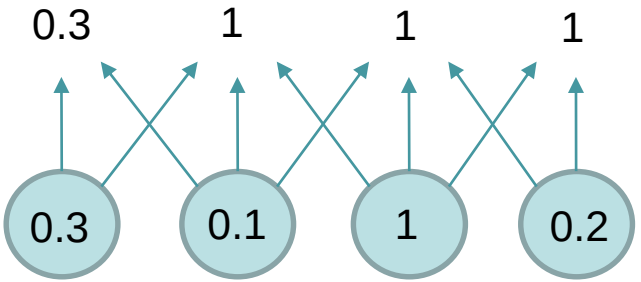
Пространственное объединение (1)

- Смысл объединения состоит в том, чтобы заменить выход сети сводной статистикой в окрестности выходов
- Примеры:
  - объединение по максимуму (max pooling),
  - усреднение по прямоугольной области (average pooling),
  - L^2-норма в прямоугольной окрестности,
  - взвешенное среднее на основании расстояния относительно центрального пикселя

Пространственное объединение (2)

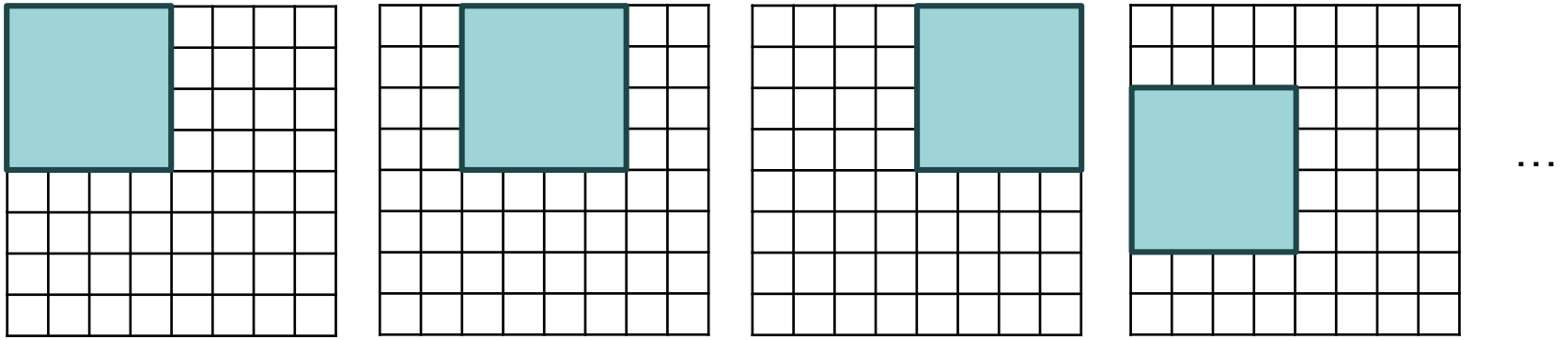
- ***Независимо от выбора функции пространственное объединение помогает сделать представление инвариантным относительно смещения входов***
- Если объект немного сместится на изображении, то значения на выходе стадии объединения практически не изменятся
- Например, если решается задача определения наличия лица на изображении, то достаточно информации о том, что на левой и правой частях лица есть по одному глазу
- Пример применения пространственного объединения по максимуму (отличия только на границах)





Пространственное объединение (3)

- ***Операция пространственного объединения обобщает отклики в некоторой окрестности, поэтому на данной стадии можно использовать меньшее количество нейронов***
- Реализуется посредством объединения окрестностей с шагом (stride), большим единицы. Пример обхода (stride=2):



- Повышается вычислительная эффективность сети, поскольку размер входа следующего слоя в несколько раз меньше (примерно в stride-раз), чем входной слой предыдущего сверточного слоя

ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ СВЕРТОЧНОЙ СЕТИ

Одномерные входные данные

Одноканальные данные	Многоканальные данные
Аудиосигнал • Дискретный сигнал, полученный с некоторым шагом по времени • Свертка вычисляется вдоль временной оси	***Данные анимации скелета*** • В каждый момент времени поза персонажа описывается углами, образованными в точках, соответствующих суставам скелета • Каждый канал данных, который подается на вход сверточной сети, представляет собой угол вокруг одной оси одного сустава

Двумерные входные данные

Одноканальные данные	Многоканальные данные
Предобработанный аудиосигнал • Сигнал после применения дискретного преобразования Фурье • Двумерная матрица, в которой строки отвечают различным частотам, столбцы – различным точкам по времени	***Цветное изображение*** • Изображение в формате RGB (или BGR) • Ядро свертки двигается в горизонтальном и вертикальном направлениях одновременно, обеспечивая тем самым инвариантность относительно операции сдвига

Трехмерные входные данные

Одноканальные данные	Многоканальные данные
Пространственные данные • Типичным примером являются данные компьютерной томографии	***Цветное видео*** • Набор двумерных цветных изображений

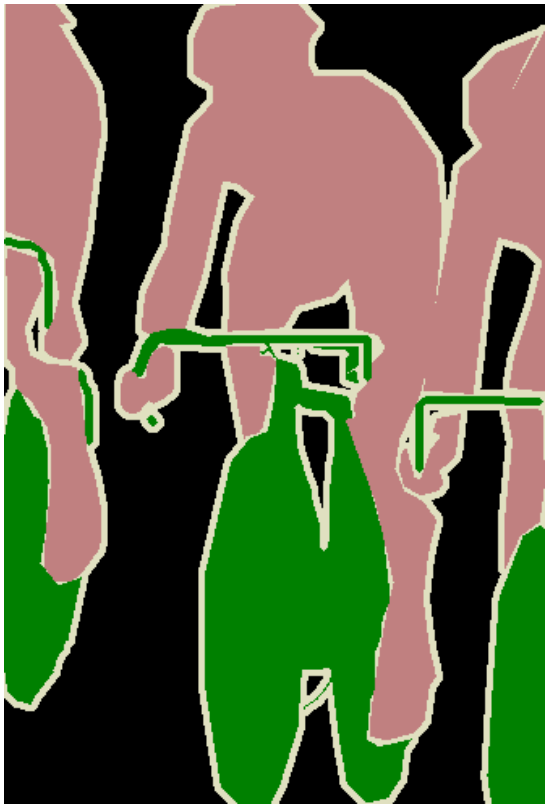
Выходные данные сверточной сети (1)

- Сверточные нейронные сети позволяют генерировать выходные данные высокой размерности
- Как правило, выход – это объект, который является тензором, полученным на выходе стандартного сверточного (или полностью связанного) слоя сети

- Например, модель может порождать трехмерный тензор с элементами $S_{i,j,k}$, отвечающими вероятности принадлежности пикселя (i, j) к классу k. В результате модель позволяет разметить каждый пиксель и выделить объекты на изображении, т.е. решить задачу семантической сегментации изображений

Выходные данные сверточной сети (2)

Оригинал



Разметка

Результат сегментации

* The PASCAL Visual Object Classes Homepage [http://host.robots.ox.ac.uk/pascal/VOC].

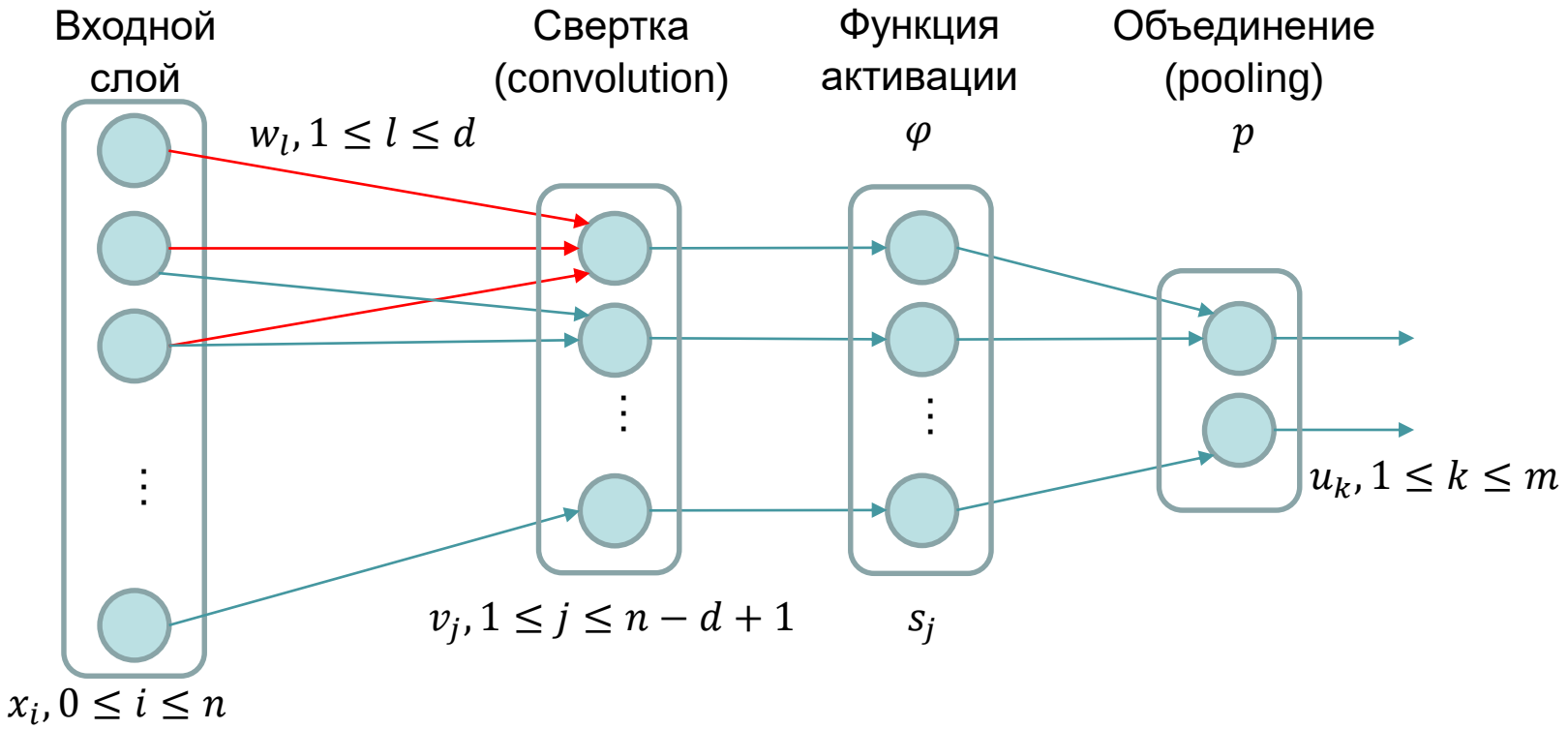
Выходные данные сверточной сети (3)

- Вид выходных данных сверточной сети зависит от ***конкретной прикладной задачи***, которая решается

МЕТОД ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ ДЛЯ СВЕРТОЧНЫХ СЕТЕЙ

Метод обратного распространения ошибки. Предположения и обозначения

- Рассмотрим сверточную сеть, содержащую только один типовой сверточный слой
- Считаем, что входные данные одномерные
- Обозначим как $E(w)$ функцию ошибки



Прямой проход. Вычисление производных (1)

- Результат применения свертки к входному сигналу выглядит следующим образом:

$$v_j = \sum_{i=1}^{d} x_{j+i-1} w_i = w^T x_{j:j+d-1}$$

- Тогда производная от функции свертки по весовым коэффициентам ядра вычисляется по формуле

$$\frac{\partial v_j}{\partial w_i} = x_{j+i-1}, \forall i = \overline{1, d}$$

Прямой проход. Вычисление производных (2)

- Производная от функции ошибки по весам ядра свертки:

$$\frac{\partial E(w)}{\partial w_i} = \frac{\partial E}{\partial v}\frac{\partial v}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial v_j} x_{j+i-1} =$$

$$= \langle \delta^{(conv)} * x \rangle(i) = \langle x * \delta^{(conv)} \rangle(i),$$

$$\delta^{(conv)} = \left(\frac{\partial E}{\partial v_j}\right)_{j=\overline{1,n-d+1}}, \qquad \frac{\partial E}{\partial w} = \langle x * \delta^{(conv)} \rangle$$

Прямой проход. Вычисление производных (3)

- Производные от функции активации и функции пространственного объединения зависят от вида функций
- В качестве функции объединения может быть выбрана любая дифференцируемая вещественнозначная функция:

$$
\begin{cases}
\dfrac{\sum_{i=1}^{q} x_i}{q}, \quad \dfrac{\partial p}{\partial x_i} = \dfrac{1}{q}, & average\ pooling \\
\max\limits_{i=1,q} x_i, \quad \dfrac{\partial p}{\partial x_i} = \begin{cases} 1, x_i = \max\limits_{i=1,q} x_i \\ 0, \text{иначе} \end{cases}, & max - pooling \\
\|x\|_p = \left(\sum_{i=1}^{q} |x_i|^p \right)^{\frac{1}{p}}, \dfrac{\partial p}{\partial x_i} = \left(\sum_{i=1}^{q} |x_i|^p \right)^{\frac{1}{p}-1} |x_i|^{p-1}, & L_p - pooling \\
p\colon \mathbb{R}^q \to \mathbb{R}, \quad \exists \dfrac{\partial p}{\partial x_i}
\end{cases}
$$

Прямой проход. Вычисление производных (4)

- Производная функции ошибки вычисляется следующим образом:

$$\frac{\partial E(w)}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial u_k} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j} x_{j+i-1} = \langle x * \delta^{(conv)} \rangle (x_i),$$

$$\delta_{j,k}^{(conv)} = \frac{\partial E}{\partial u_k} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j} = \delta_k^{(pool)} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j}, \qquad \delta_k^{(pool)} = \frac{\partial E}{\partial u_k}$$

Обратный проход. Коррекция весов сети

- Выбор типа пространственного объединения определяет, в каком направлении распространяется ошибка
- В случае выбора объединения по максимуму очевидно, что градиент функции ошибки на последнем слое идет в сторону нейрона s_j, на выходе которого получено максимальное значение
- В общем случае градиент распространяется в соответствии с значением $\delta_k^{(pool)}, 1 \le k \le m$
- Это значение передается в обратном направлении и значение градиентов на сверточном слое определяется в соответствии с формулой для вычисления $\delta_{j,k}^{(conv)}, 1 \le j \le n - d + 1$

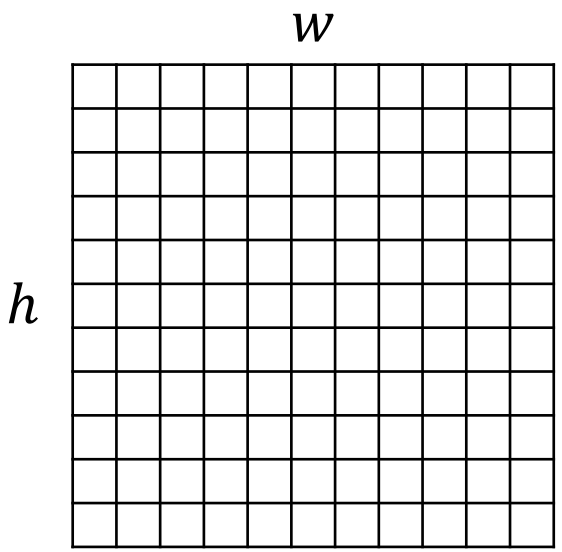
ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА ПАРАМЕТРОВ СЕТИ. ОЦЕНКА ОБЪЕМА ПАМЯТИ

Зачем определять количество параметров сети?

- Позволяет ***оценить размерность пространства параметров***, в котором решается задача минимизации целевой функции
- Позволяет ***оценить объем памяти***, необходимой для обучения/тестирования нейронной сети

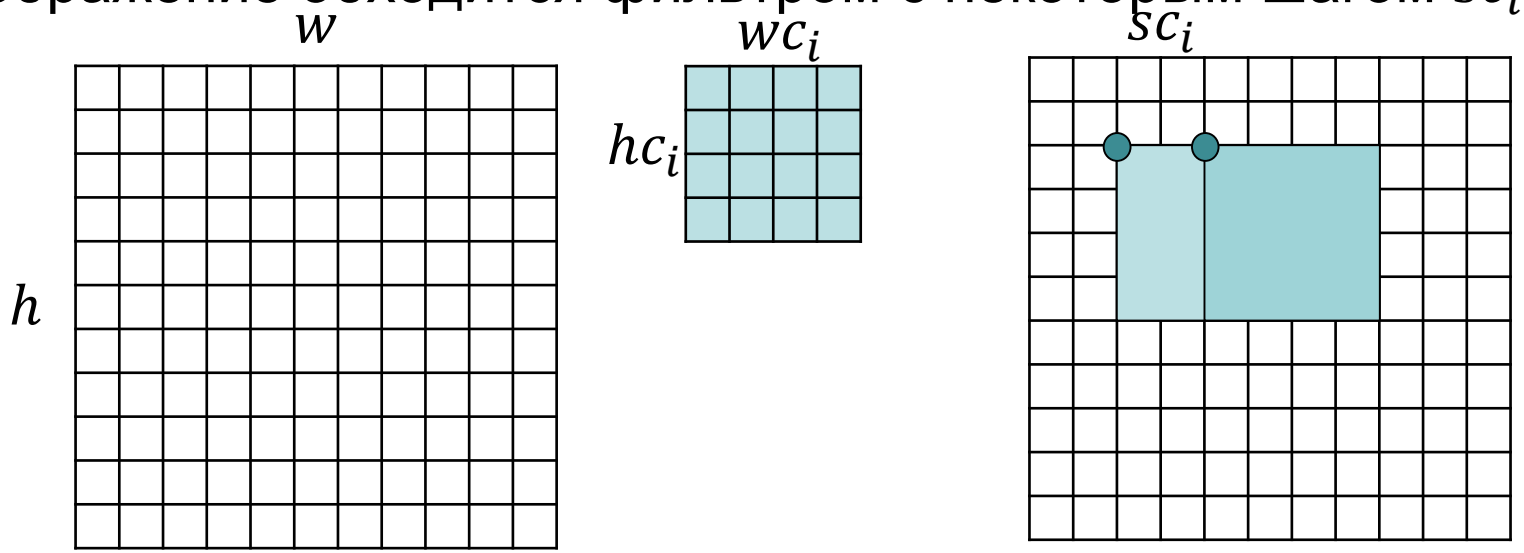
Определение количества параметров сверточной сети (1.1)

- Предположим, что на входе сети имеется одноканальное изображение разрешения $w \times h$



Определение количества параметров сверточной сети (1.2)

- Сеть содержит N сверточных слоев (троек, состоящих из свертки, вычисления функции активации и пространственного объединения)
- На каждом слое выполняется свертка карты признаков с фильтром, размер ядра которого составляет $wc_i \times hc_i$, где $1 \leq i \leq N$ – номер слоя
- Изображение обходится фильтром с некоторым шагом sc_i



Определение количества параметров сверточной сети (1.3)

- Количество обучаемых параметров сети:

$$\sum_{i=1}^{N} wc_i \cdot hc_i$$

Определение количества параметров сверточной сети (2)

- В общем случае на каждом слое может вычисляться свертка не с одним ядром, а с несколькими ядрами одинакового размера
- В предположении, что на слое с номером i выполняется k_i сверток, суммарное количество параметров составляет

$$\sum_{i=1}^{N} k_i \cdot wc_i \cdot hc_i$$

Объем памяти, необходимый для хранения сверточной нейронной сети (1)

- Входной слой сети содержит $w \times h$ пикселей
- Применение k_1 сверток к входному слою приводит к формированию карты признаков размерности

$$k_1 \times \left(\left[\frac{w - wc_1}{sc_1}\right] + 1\right) \times \left(\left[\frac{h - hc_1}{sc_1}\right] + 1\right)$$

- Применение функции активации к каждому элементу полученной карты дает карту признаков идентичной размерности

$$k_1 \times \left(\left[\frac{w - wc_1}{sc_1}\right] + 1\right) \times \left(\left[\frac{h - hc_1}{sc_1}\right] + 1\right)$$

Объем памяти, необходимый для хранения сверточной нейронной сети (2)

- Введем обозначения $w_1 = \left[\frac{w-wc_1}{sc_1}\right] + 1$, $h_1 = \left[\frac{h-hc_1}{sc_1}\right] + 1$
- Допустим, что операция пространственного объединения применяется к каждому каналу карты признаков (выход функции активации)
- Размер рассматриваемой окрестности составляет $wp_1 \times hp_1$, и проход осуществляется с шагом sp_1
- Тогда размерность карты признаков после операции объединения

$$k_1 \times \left(\left[\frac{w_1 - wp_1}{sp_1}\right] + 1\right) \times \left(\left[\frac{h_1 - hp_1}{sp_1}\right] + 1\right)$$

Объем памяти, необходимый для хранения сверточной нейронной сети (3)

- Объем памяти, необходимый для хранения первого сверточного слоя определяется формулой:

$$\left(2k_1w_1h_1 + k_1\left(\left[\frac{w_1 - wp_1}{sp_1}\right] + 1\right)\left(\left[\frac{h_1 - hp_1}{sp_1}\right] + 1\right)\right) \cdot sizeof(type)$$

- Если имеется N сверточных слоев, устроенных аналогичным образом, то для хранения всей сети требуется следующий объем памяти:

$$\left(w \cdot h + \sum_{i=1}^{N}\left(2k_iw_ih_i + k_i\left(\left[\frac{w_i - wp_i}{sp_i}\right] + 1\right)\left(\left[\frac{h_i - hp_i}{sp_i}\right] + 1\right)\right)\right) \cdot sizeof(type),$$

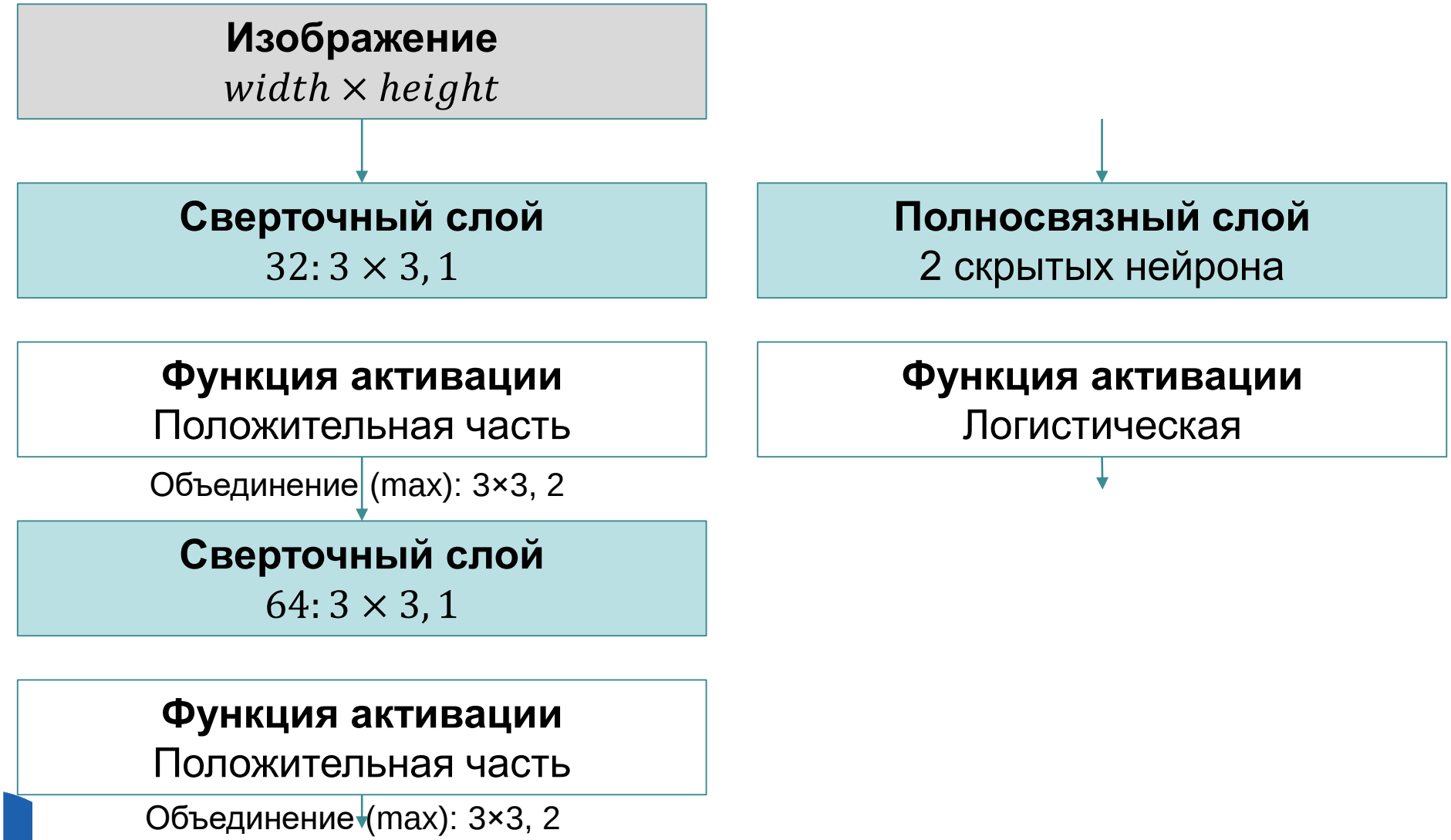
$$w_i = \left[\frac{w_{i-1} - wc_i}{sc_i}\right] + 1, h_i = \left[\frac{h_{i-1} - hc_i}{sc_i}\right] + 1, \qquad w_0 = w, h_0 = h$$

Объем памяти, необходимый для хранения сверточной нейронной сети (4)

- Рассмотренный пример является частным случаем сверточной сети и демонстрирует лишь общую процедуру оценки объема памяти, необходимой для ее хранения
  - На входе сети может быть сигнал, отличный по структуре от рассмотренного (многомерный сигнал)
  - При выполнении операции свертки входная карта признаков может дополняться полями с целью сохранения размерности карты на выходе
  - Сверточная сеть наряду со сверточными слоями может содержать полностью связанные слои
  - В процессе обучения на вход сети могут подаваться изображения целым набором фиксированного размера – ***пачкой*** (batch) – с целью повышения эффективности вычислений и скорости обучения сети

ПРИМЕР ПОСТРОЕНИЯ СВЕРТОЧНОЙ СЕТИ ДЛЯ РЕШЕНИЯ ЗАДАЧИ

Архитектура сверточной сети



Пример применения сверточных сетей к задаче предсказания пола человека по фотографии

```
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)
```

Тестовая инфраструктура

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Инструменты:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Результаты экспериментов

Название	Структура сети	Параметры обучения	Точность, %	Время обучения, с
CNN-1	Conv((3,3,32),0,Rectlin), Pooling((3,3),2,'max'), Conv((3,3,64),0,Rectlin), Pooling((3,3),2,'max'), Affine(2,0), Logistic)	batch_size = 128 epoch_count = 30 backend = gpu GradientDescentMomentum(0.01, momentum_coef= 0.9, wdecay=0.0005)	79.3	1582
CNN-2	Conv((3,3,32),Explin), Pooling((5,5),2,'max'), Conv((3,3,32),Explin), BatchNorm, Conv((1,1,64),Explin), BatchNorm, Conv((3,3,64),Explin), BatchNorm, Conv((1,1,128),Explin), BatchNorm, Conv((3,3,128),Explin), BatchNorm, Conv((3,3,2),Explin), BatchNorm, Conv((1,1,2),0,Explin), BatchNorm, Pooling('avg'), Affine(2,0,Logistic)		**83.5**	2030

Лучшие результаты для многослойных полносвязных и сверточных сетей

Название	Структура сети	Точность, %	Время обучения, с
FCNN-1	Affine(128,0,Tanh), Affine(2,0,Logistic)	71.2	932
FCNN-2	Affine(128,0,Tanh), Affine(64,0,Tanh), Affine(2,0,Logistic)	73.5	977
FCNN-3	Affine(400,0,Rectlin), Affine(50,0,Logistic), Affine(2,0,Logistic)	**77.7**	1013
CNN-1	Conv((3,3,32),0,Rectlin), Pooling((3,3),2,'max'), Conv((3,3,64),0,Rectlin), Pooling((3,3),2,'max'), Affine(2,0), Logistic)	79.3	1582
CNN-2	Conv((3,3,32),Explin), Pooling((5,5),2,'max'), Conv((3,3,32),Explin), BatchNorm, Conv((1,1,64),Explin), BatchNorm, Conv((3,3,64),Explin), BatchNorm, Conv((1,1,128),Explin), BatchNorm, Conv((3,3,128),Explin), BatchNorm, Conv((3,3,2),Explin), BatchNorm, Conv((1,1,2),0,Explin), BatchNorm, Pooling('avg'), Affine(2,0,Logistic)	**83.5**	2030

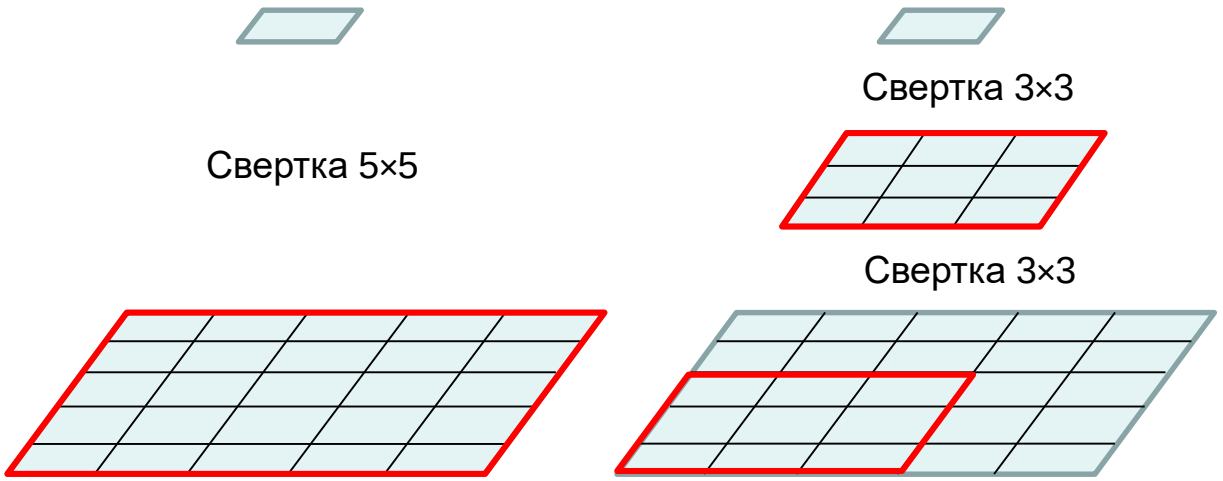
ПРИНЦИПЫ ПОСТРОЕНИЯ СВЕРТОЧНЫХ СЕТЕЙ

Принципы построения сверточных сетей (1)

- ***Выполнять предварительную обработку входных данных***
  - Вычитание среднего изображения, полученного по всем изображения тренировочного множества
  - Центрирование изображений
- ***Избегать «узких горлышек» в представлении сети, особенно на начальных слоях***
  - При построении архитектуры сети имеет смысл избегать экстремального сжатия информации
  - Размерность представления дает приблизительную оценку информационного содержания контента

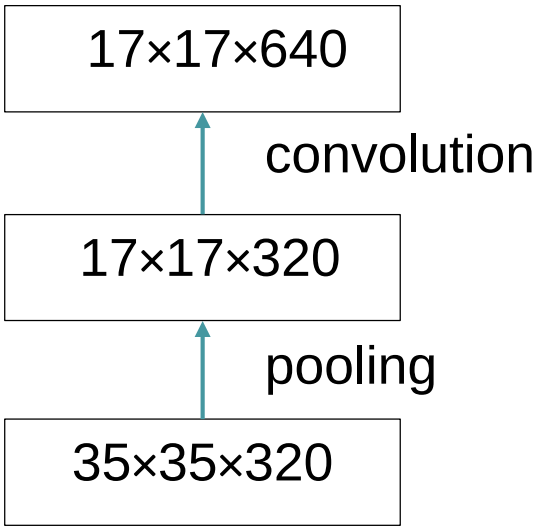
Принципы построения сверточных сетей (2)

- ***Заменять сверточные слои большой размерности стеком сверток более низкой размерности***
  - Свертку с фильтром 5×5 можно заменить двумя последовательными свертками с фильтрами размера 3×3
  - При этом формируется сеть с меньшим числом параметров, но с тем же размером входа и глубиной выхода



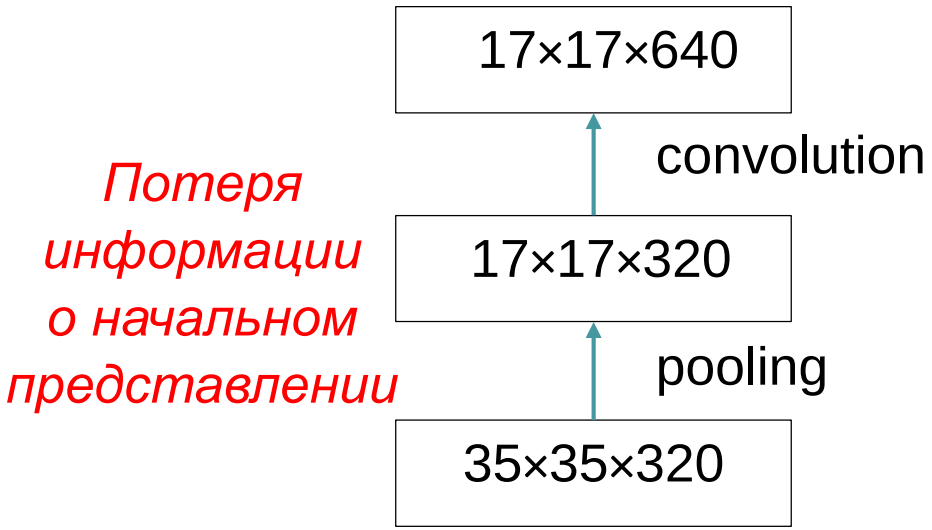
Принципы построения сверточных сетей (3.1)

- ***Пространственную агрегацию следует выполнять по картам признаков более низкой размерности для снижения вычислительной сложности***
  - Реализуется посредством пространственного объединения (pooling) или введения inception-модулей



Принципы построения сверточных сетей (3.1)

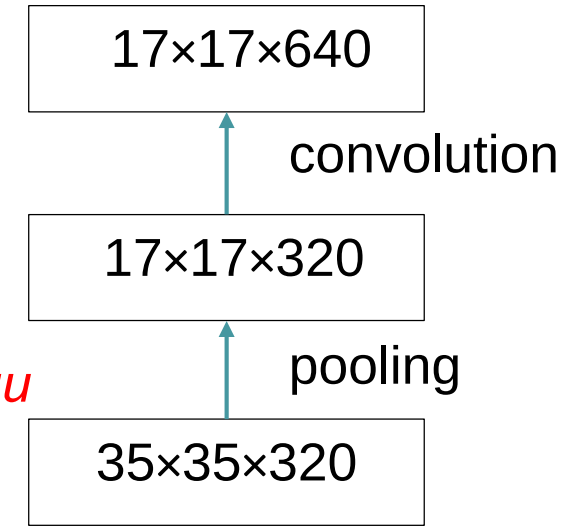
- ***Пространственную агрегацию следует выполнять по картам признаков более низкой размерности для снижения вычислительной сложности***
  - Реализуется посредством пространственного объединения (pooling) или введения inception-модулей

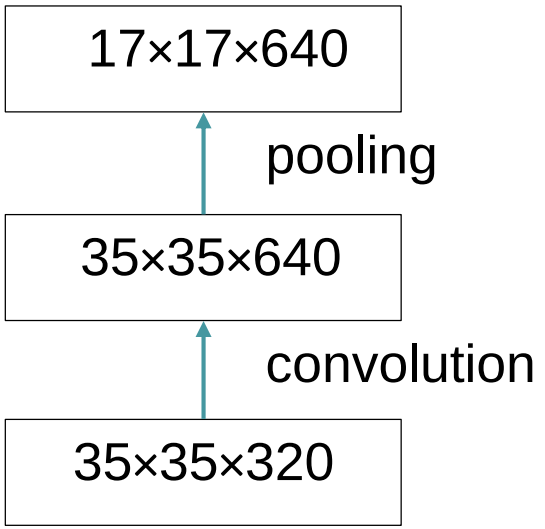


Принципы построения сверточных сетей (3.1)

- ***Пространственную агрегацию следует выполнять по картам признаков более низкой размерности для снижения вычислительной сложности***
  - Реализуется посредством пространственного объединения (pooling) или введения inception-модулей

Потеря информации о начальном представлении



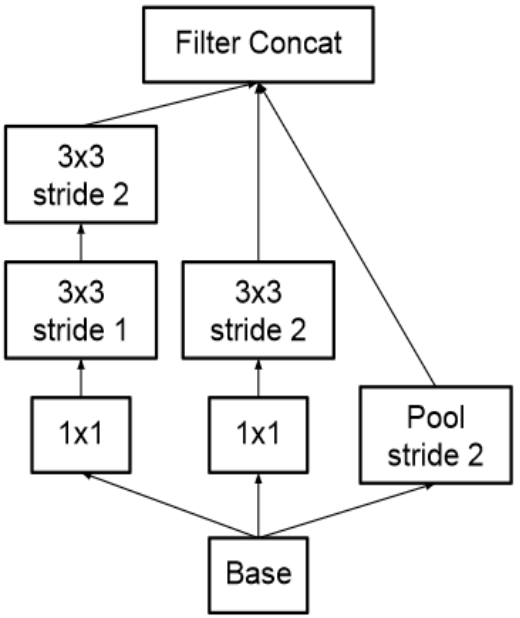


Принципы построения сверточных сетей (3.1)

- ***Пространственную агрегацию следует выполнять по картам признаков более низкой размерности для снижения вычислительной сложности***
  - Реализуется посредством пространственного объединения (pooling) или введения inception-модулей

Принципы построения сверточных сетей (3.2)

- ***Пространственную агрегацию следует выполнять по картам признаков более низкой размерности для снижения вычислительной сложности***
  - Реализуется посредством пространственного объединения (pooling) или введения inception-модулей



* Szegedy C., Vanhoucke V., Ioffe S., Shlens J. Rethinking the Inception Architecture for Computer Vision – [https://arxiv.org/pdf/1512.00567v3.pdf].

Принципы построения сверточных сетей (4)

- ***Сбалансировать глубину и ширину сети***
  - Увеличение ширины и глубины сети может способствовать созданию сетей более высокого качества
  - Оптимальная производительность сети может быть достигнута путем балансировки количества фильтров на каждом сверточном слое и глубины сети

ПРОБЛЕМА ДЕГРАДАЦИИ МОДЕЛИ. ГЛУБОКИЕ ОСТАТОЧНЫЕ СЕТИ

Проблема деградация модели

- ***Проблема деградации глубокой модели:***
  - С ростом глубины сети точность насыщается и затем быстро начинает уменьшаться (деградировать)

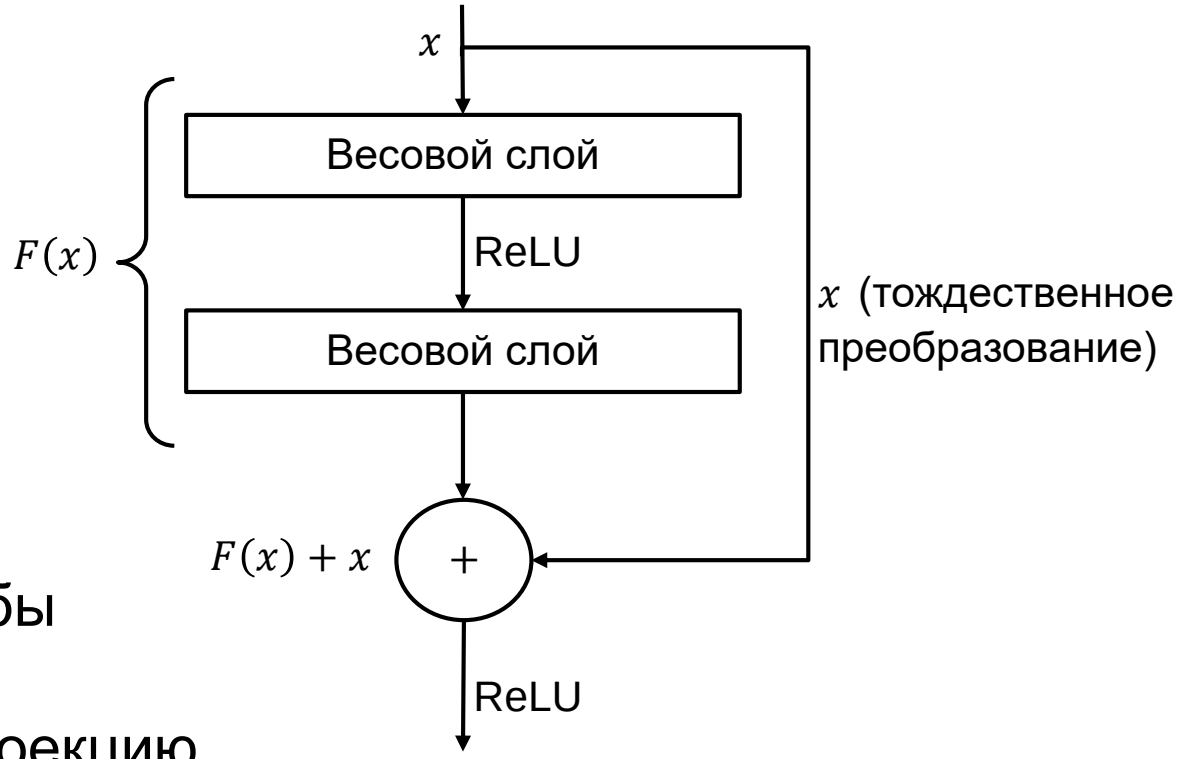
- Проблема не является следствием переобучения модели
- Добавление дополнительных слоев приводит к еще большему значению тренировочной ошибки
- Деградация точности обучения указывает на то, что не все глубокие модели одинаково легко оптимизируются

Глубокие остаточные сети (1)

- Вместо того, чтобы предполагать, что некоторая последовательность слоев сети напрямую аппроксимирует базовое отображение, считается, что эти слои аппроксимируют остаточное отображение
- Введем обозначения
  - $H(x)$ – базовое отображение
  - $F(x) = H(x) - x$ – остаточное отображение
  - Базовое отображение можно представить как поэлементное сложение карт признаков $F(x) + x$
- Предполагается, что остаточное отображение проще оптимизировать по сравнению базовым. В крайнем случае, если тождественное преобразование является оптимальным, то проще остаток свести к нулю, чем аппроксимировать тождественное отображение набором нелинейных слоев

Глубокие остаточные сети (2)

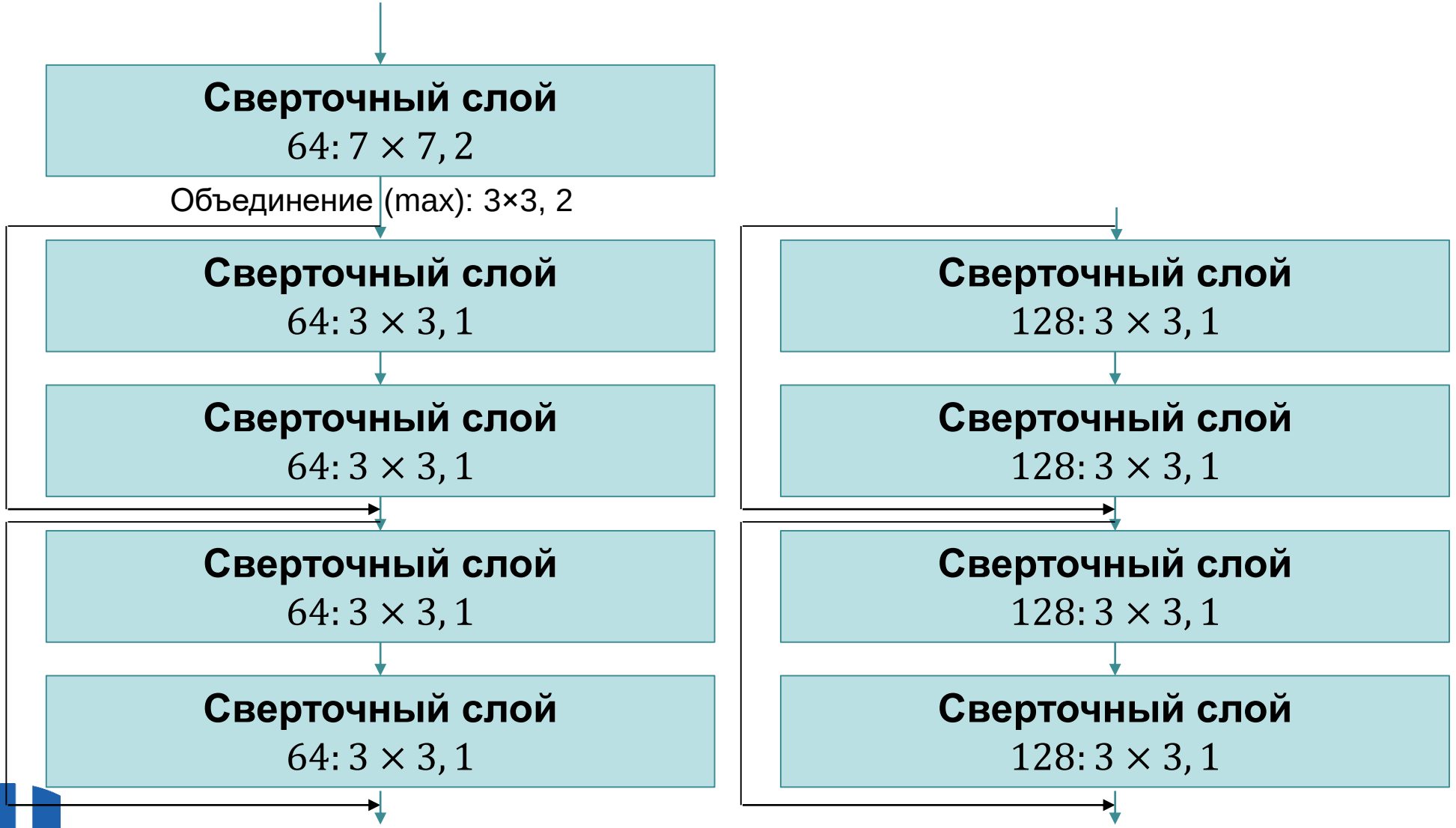
- Отображение $F(x) + x$ можно представить с помощью сети прямого распространения с быстрыми связями (shortcut connections)



- Для показанного примера $y = F(x, W_i) + x = W_2\varphi(W_1 x) + x$, где $\varphi(\cdot)$ – функция активации ReLU
- $F(x, W_i)$ и x могут иметь разную размерность, чтобы исправить эту ситуацию достаточно выполнить проекцию входного вектора признаков $y = F(x, W_i) + W_s x$

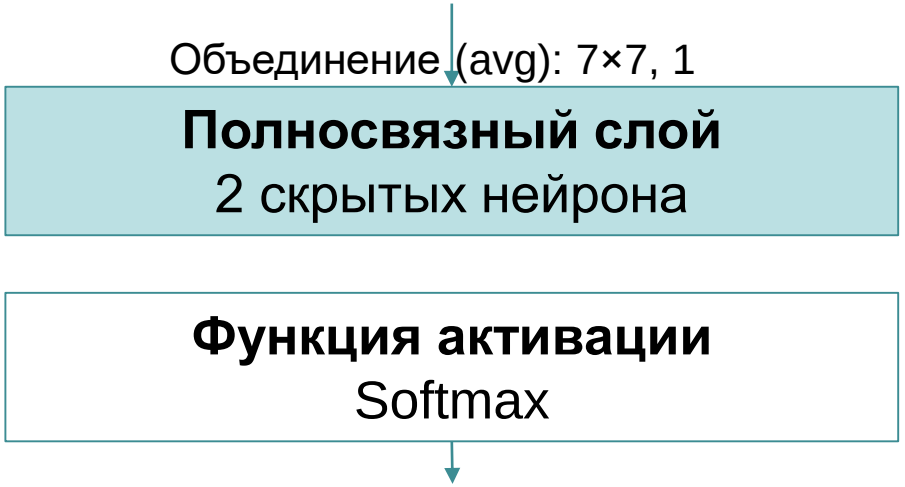
ПРИМЕР ПОСТРОЕНИЯ ОСТАТОЧНОЙ СЕТИ ДЛЯ РЕШЕНИЯ ЗАДАЧИ

Архитектура остаточной сети ResNet-18 (1)



Архитектура остаточной сети ResNet-18 (2)

Архитектура остаточной сети ResNet-18 (3)



- ***Примечания:***
  - После каждого сверточного слоя следует нормализация **`BatchNorm`** и применение функции активации **`ReLU`**
  - ResNet-34, ResNet-50, ResNet-101 строятся аналогичным образом

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (1)

```
def generate_resnet18_model():
    layers = make_resnet_base([2, 2, 2, 2], block='basic')
    layers.append(BatchNorm())
    layers.append(Activation(Rectlin()))
    layers.append(Pooling('all', op='avg'))
    layers.append(Affine(2, init=Kaiming(local=False),
                         activation=Softmax()))

    model = Model(layers=layers)

    cost = GeneralizedCost(costfunc=CrossEntropyMulti())

    return (model, cost)
```

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (2)

```
def make_resnet_base(stage_depths, block='basic',
                     base_channels=64):
    from math import log2

    def conv_params(fsize, nfm, stride=1, relu=True,
                    batch_norm=True):
        return dict(
            fshape=(fsize, fsize, nfm),
            strides=stride,
            padding=fsize // 2,
            activation=(Rectlin() if relu else None),
            init=Kaiming(local=True),
            batch_norm=batch_norm)
...
```

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (3)

```
def module_basic(nfm, first=False, stride=1):
        # building block for ResNet-18
        mainpath = [] if first else [BatchNorm(),
                                     Activation(Rectlin())]
        mainpath.append(Conv(**conv_params(3, nfm,
                             stride=stride)))
        mainpath.append(Conv(**conv_params(3, nfm,
                             relu=False, batch_norm=False)))
        sidepath = Conv(**conv_params(1, nfm, stride=stride,
                             relu=False, batch_norm=False))
                   if (first or (stride != 1)) else SkipNode()

        return MergeSum([sidepath, mainpath])
...
```

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (4)

```
def module_bottleneck(nfm, first=False, stride=1):
        # building block for ResNet-50, -101, -152
        mainpath = [] if first else [BatchNorm(),
                                     Activation(Rectlin())]
        mainpath.append(Conv(**conv_params(1, nfm,
                                     stride=stride)))
        mainpath.append(Conv(**conv_params(3, nfm)))
        mainpath.append(Conv(**conv_params(1, nfm * 4,
                                     relu=False, batch_norm=False)))

        sidepath = Conv(**conv_params(1, nfm * 4,
                stride=stride, relu=False, batch_norm=False))
                if (first or (stride != 1)) else SkipNode()
        return MergeSum([sidepath, mainpath])
```

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (5)

```
blocks = {'basic': module_basic, 'bottleneck':
           module_bottleneck}
# 18:  [2, 2, 2, 2], output = 512, block = 'basic'
# 34:  [3, 4, 6, 3], output = 512, block = 'basic'
# 50:  [3, 4, 6, 3], output = 2048, block = 'bottleneck'
stage_depths_populated = []
for stage, depth in enumerate(stage_depths):
    stage_depths_populated.extend([stage] * depth)
# nfms is a list of channel counts in blocks
nfms = [2**(stage + int(log2(base_channels))) for stage
       in stage_depths_populated]
strides = [1 if cur == prev else 2 for cur, prev
           in zip(nfms[1:], nfms[:-1])]
module = blocks[block]
```

Пример применения остаточных сетей к задаче предсказания пола человека по фотографии (6)

```
    # Now construct the network
    layers = [
        Conv(**conv_params(7, base_channels, 2)),
        Pooling(fshape=(3, 3), padding=1, strides=2,
        op='max')
    ]
    layers.append(module(nfms[0], first=True))
    for nfm, stride in zip(nfms[1:], strides):
        layers.append(module(nfm, stride=stride))

    return layers
```

Тестовая инфраструктура

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Инструменты:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Лучшие результаты для многослойных полносвязных, сверточных и остаточных сетей

Название	Точность, %	Время обучения, с
FCNN-1	71.2	932
FCNN-2	73.5	977
FCNN-3	**77.7**	1013
CNN-1	79.3	1582
CNN-2	**83.5**	2030
ResNet-18 (90 эпох)	**81.3**	15127
ResNet-50 (30 эпох)	**80.9**	11849

Заключение

- Применение сверточных сетей позволяют повысить точность решения задачи
- Построение остаточных нейронных сетей не позволяет получить существенного выигрыша относительно полносвязных моделей в данной задаче
- Эффективность применения остаточных сетей при решении других задач необходимо проверять экспериментально

Основная литература

- Хайкин С. Нейронные сети. Полный курс. – М.: Издательский дом «Вильямс». – 2006. – 1104 с.
- Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика. – 2002. – 344 с.
- Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Авторский коллектив

- **Кустикова Валентина Дмитриевна**
  к.т.н., ст.преп. каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  valentina.kustikova@itmm.unn.ru
- **Жильцов Максим Сергеевич**
  магистрант каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  zhiltsov.max35@gmail.com
- **Золотых Николай Юрьевич**
  д.ф.-м.н., проф. каф. АГДМ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  nikolai.zolotykh@itmm.unn.ru