Занятие 6. Сверточные сети и их основные компоненты

Гирдюк Дмитрий Викторович

29 марта 2025

СПбГУ, ПМ–ПУ, ДФС

Почему не полносвязные слои?

- На предыдущей практике мы осознали, что работа с изображениями путем применения полносвязных слоев над их векторизованными представлениями – это путь в никуда
- Мало того что параметров на это дело надо невероятно много, так и топологическая структура данных никак не учиытвается
- Нужны альтернативные способы работы с подобными данными

Инвариантность и эквивариантность/ковариантность

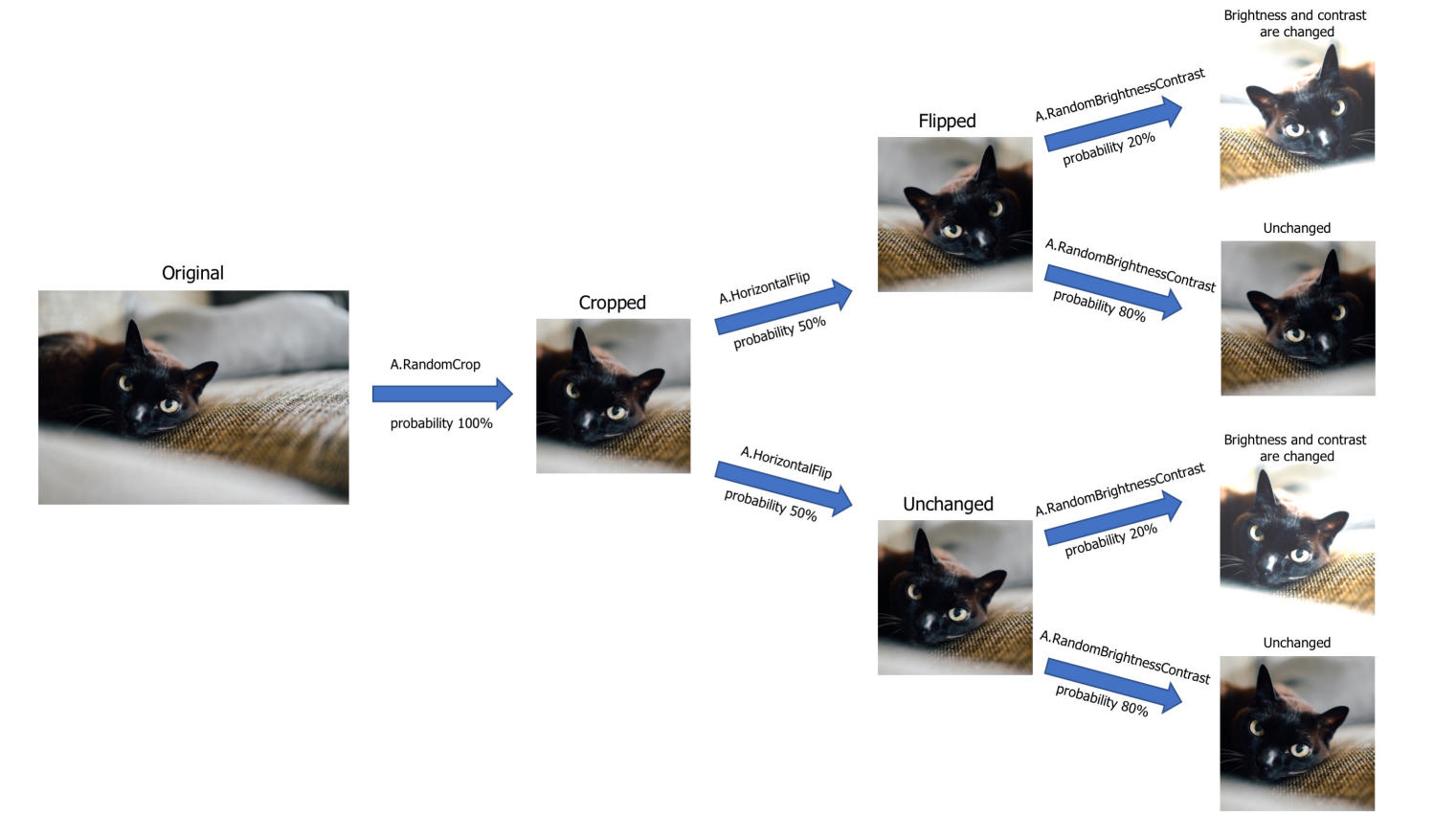
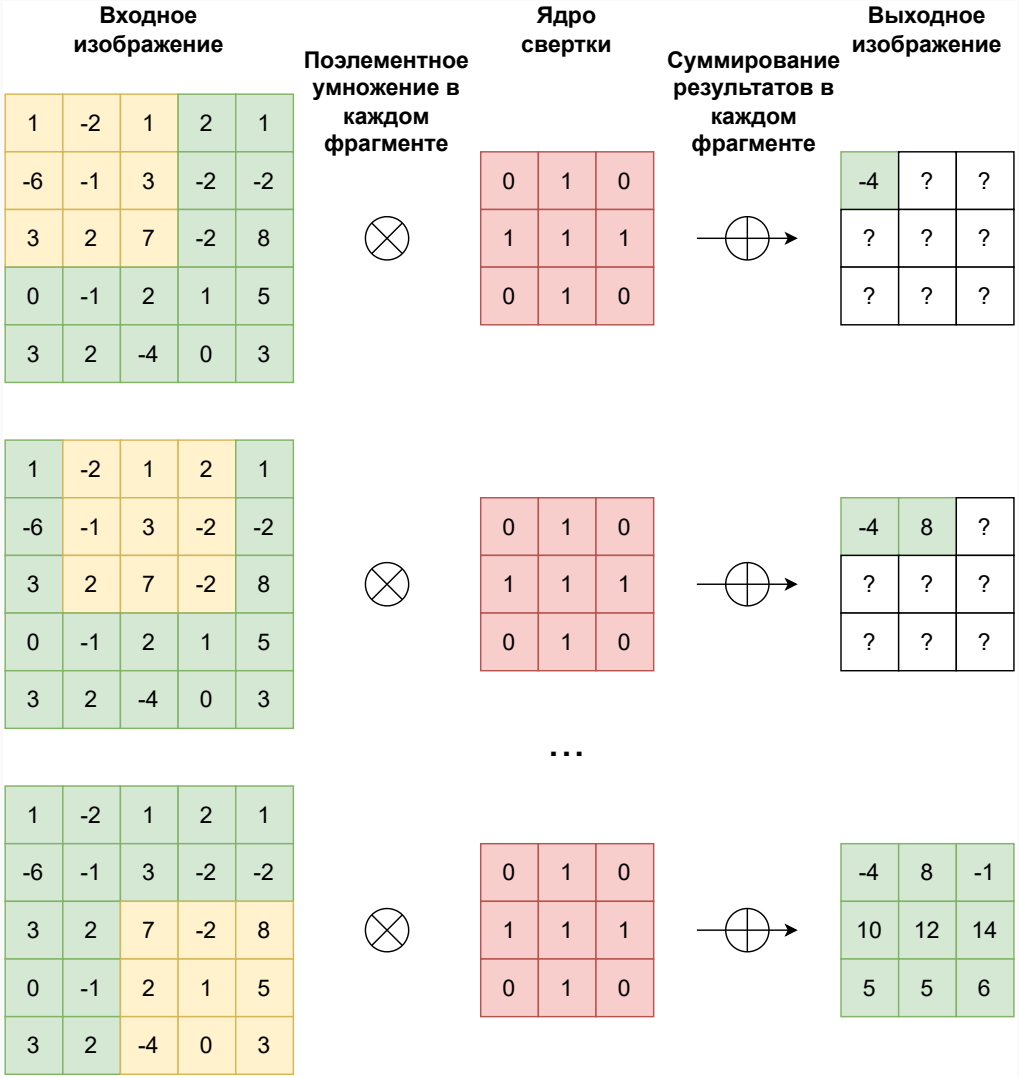
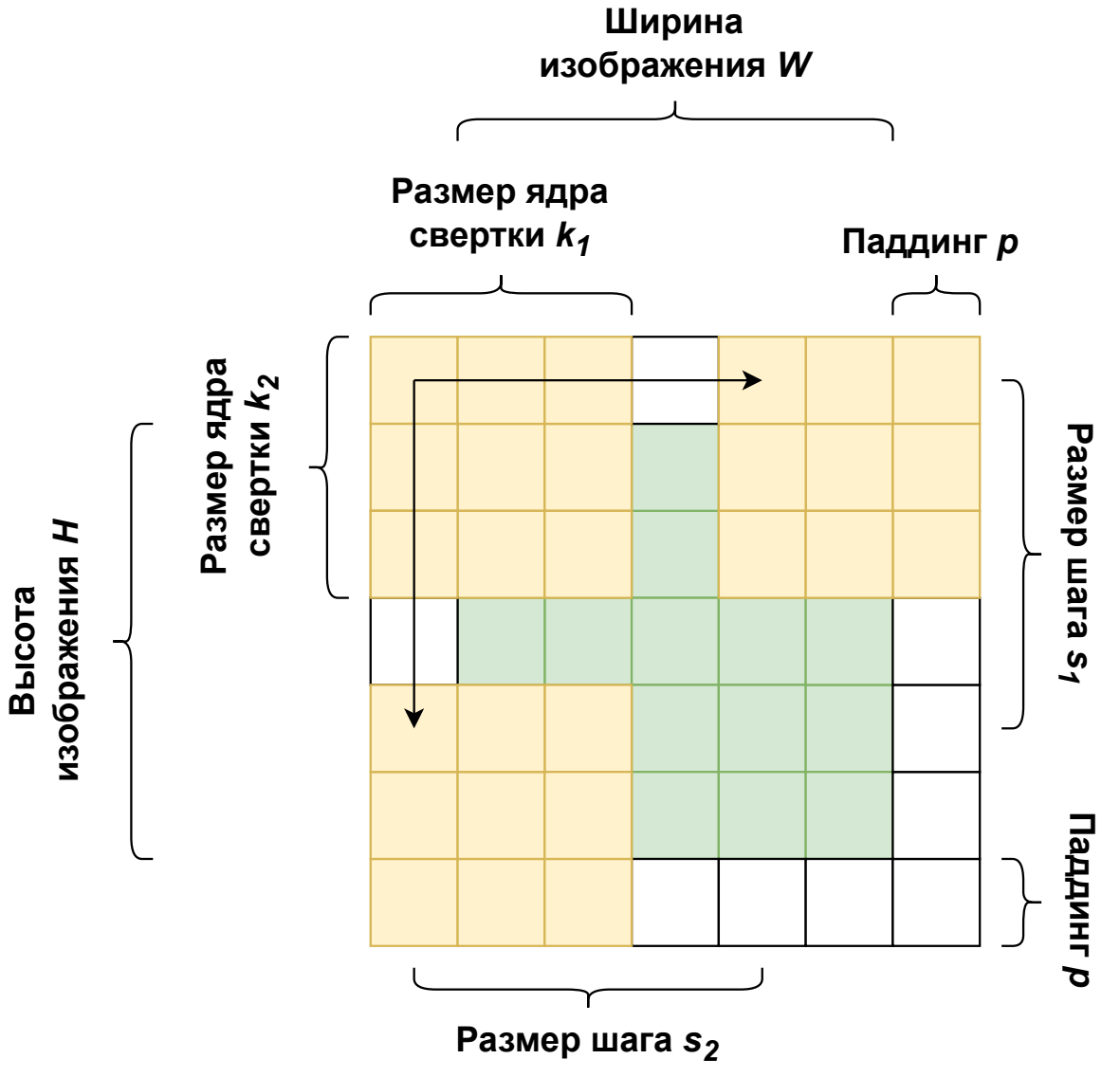


Рис. 1: Сети для работы с изображениями должны быть инвариантны и ковариантны к их всевозможным трансформациям. Картинка из [1]

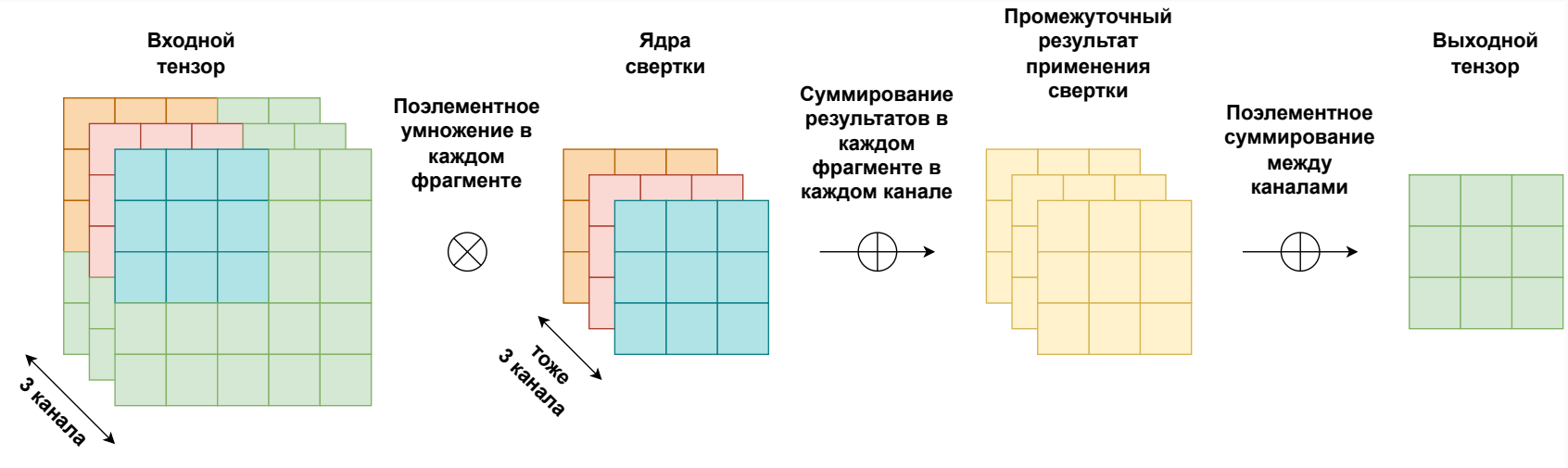
Свертки: общая идея



Свертки: основные обозначения



Свертки: многоканальный сценарий



Свертка: формальное определение

- Пусть имеем на входе трехмерный тензор $X_{H \times W \times C_{in}}$ карт признаков
- Свертка $(X * K)$, где $K_{k \times k \times C_{in}}$ есть набор из C_{in} ядер размером $k \times k$, представляет собой линейное преобразование карт признаков
- Каждое ядро свертки последовательно проходит по соответствующему ей одноканальному изображению размера $H \times W$ и поэлементно умножает веса на значения карты признаков в окне, после чего суммирует их в рамках окна
- Наконец, в получившемся наборе промежуточных карт признаков размера $\widehat{H} \times \widehat{W} \times C_{in}$ производится поэлементное суммирование между каналами, что дает на выходе новую карту признаков размера $\widehat{H} \times \widehat{W} \times 1$

Свертка: формальное определение II

- Данный подход позволяет обработать каждую часть изображения идентичным образом. По сути решаем проблему инвариантности и эквивариантности для сдвига изображения
- А теперь обобщим, сгруппируем не одну свертку, а C_{out}, а также добавим смещение для каждого выходного канала:

$$Y_{\widehat{H}\times\widehat{W}\times C_{out}} = \{(X * K)(i,j,c)\}_{i,j,c} = \\ = \sum_{l=1}^{C_{in}} \sum_{m=1}^{k} \sum_{n=1}^{k} X(i+m, j+n, l) \cdot K^c(m,n,l) + b_c$$

- Вопрос: сколько выходит обучаемых параметров?

Свертка: матричное представление

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} \sim$$

$$\sim \begin{bmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} =$$

$$= \begin{bmatrix} k_1x_1 + k_2x_2 + k_3x_4 + k_4x_5 \\ k_1x_2 + k_2x_3 + k_3x_5 + k_4x_6 \\ k_1x_4 + k_2x_5 + k_3x_7 + k_4x_8 \\ k_1x_5 + k_2x_6 + k_3x_8 + k_4x_9 \end{bmatrix}$$

Свертки: подсчет выходного разрешения

- Как мы заметили ранее, различные значения k, s и p могут давать на выходе карты признаков различного размера $\widehat{H} \times \widehat{W}$
- Формула для размера выходной карты признаков:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- Подробно про арифметику сверточных слоев можно почитать вот тут [2]

Сверточные сети

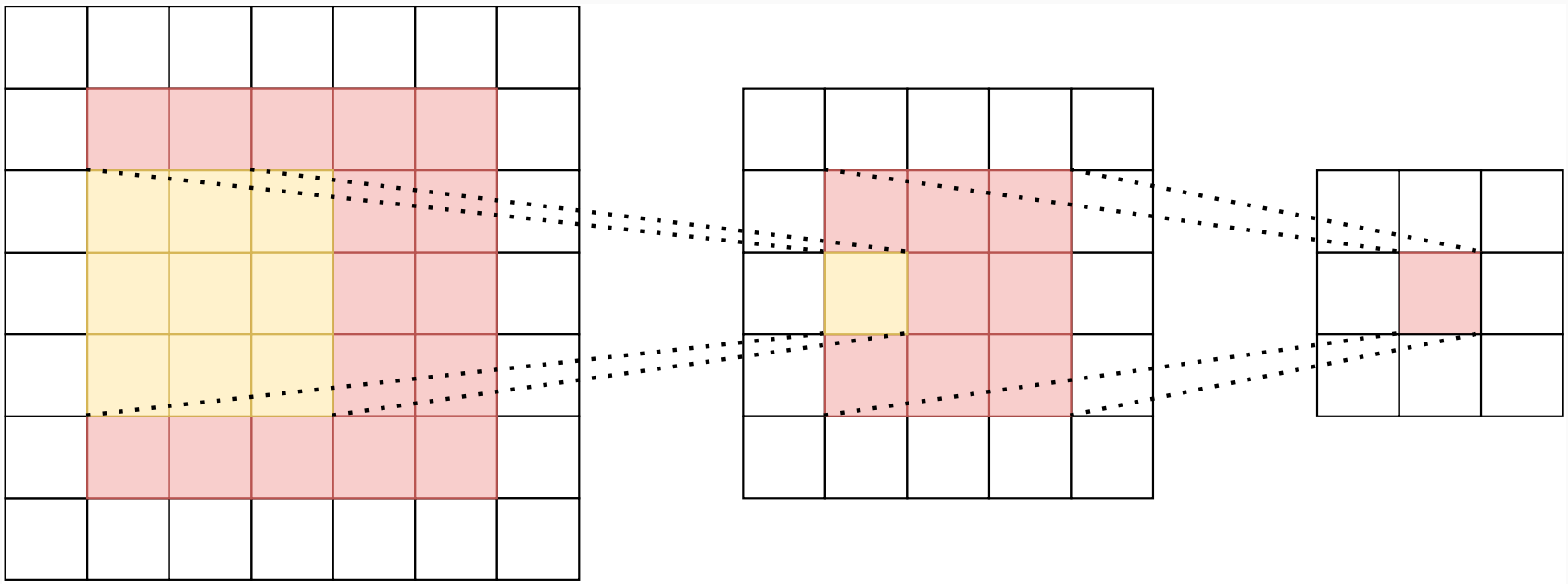
- Одного слоя сверток, конечно, недостаточно. Мы ограничиваем сеть работой в локальной окрестности размера $k \times k$. И если k достаточно большое, то мы недалеко ушли от полносвязных слоев
- Впрочем, никто не мешает нам поддерживать значение k достаточно малым (обычно нечетные и из интервала $[3, 11]$) и применять свертки поверх других сверток
- Добавляем к этому нелинейность (в виде все той же ReLU), и основной блок сверточной сети готов

Receptive field

- Хорошо, имеем последовательность сверток (не обязательно с одинаковыми k, s и p), трансформирующих разрешение карт признаков
- Одна из ключевых характеристик сверточных архитектур – receptive field, который определяется как размер области входных данных, создающей признак (скаляр) на карте признаков
- Важно при создании сверточной сети контролировать его значение для выходного сверточного слоя

Receptive field II

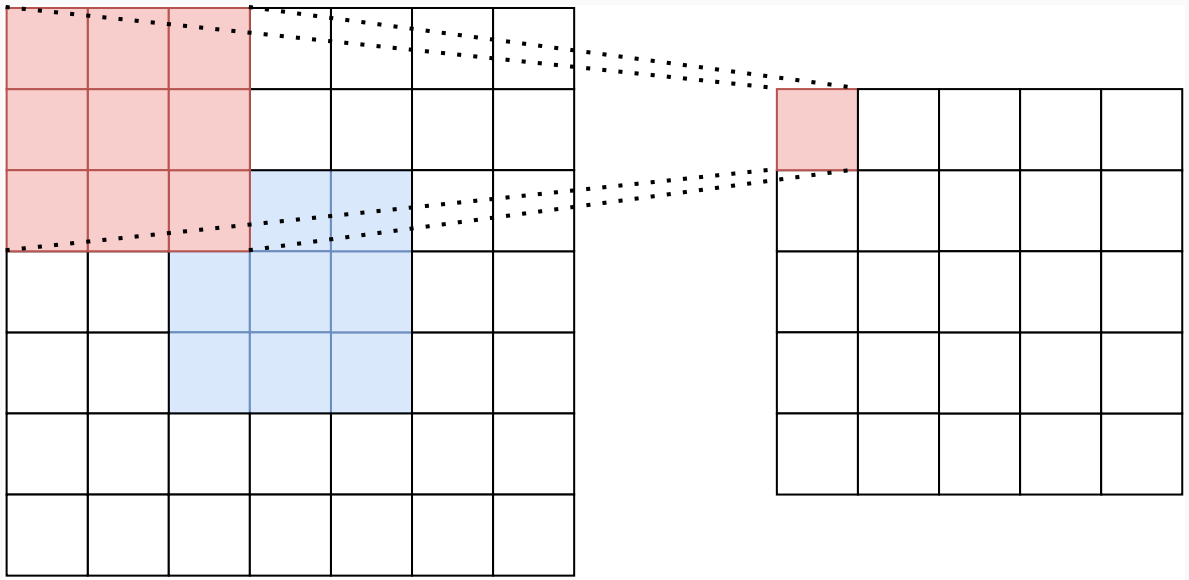
- На изображении пример последовательного применения двух сверток 3×3
- Статья по вычислению receptive field в сверточных сетях [3]



Специальные типы сверток: transposed

- В свертках мы обычно не увеличиваем разрешение выходной карты признаков
- Тем не менее, есть ряд ситуаций, когда нам необходимо (например, задачи семантической сегментации, с которыми познакомимся познее)
- Транспонированная свертка – способ получения выходной карты признаков, чье разрешение будет больше входного

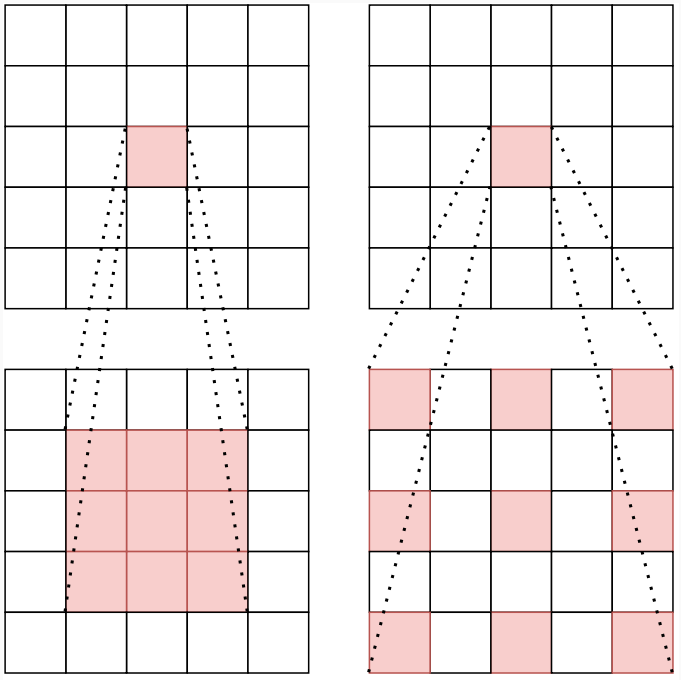
Специальные типы сверток: transposed II



Специальные типы сверток: dilated

- Dilated convolution – "свертки с дырами" , позволяют увеличивать receptive field без увеличения числа параметров:

$$o = \left\lfloor \frac{i + 2p - k - (k-1)(d-1)}{s} \right\rfloor + 1$$



Специальные типы сверток: depthwise

- Depthwise convolution – декомпозиция свертки: сначала сворачивает каждый входной канал соответствующим 2d-фильтром, а затем трансформирует C_{in} входных каналов в C_{out} выходных, используя свертку 1×1
- Пример. $12 \times 12 \times 3$ на входе, применяем свертку $5 \times 5 \times 1 \times 1$ и получаем на выходе $8 \times 8 \times 3$, а затем вторая свертка $1 \times 1 \times 3 \times 256$, которая дает окончательно тензор $8 \times 8 \times 256$

Свертки: вычисление на практике

- Существует достаточно много подходов для реализации операции свертки
- Наивный подсчет по формуле в цикле вряд ли даст хорошую производительность
- Можно использовать теорему о свертке: преобразование Фурье свёртки двух функций (или сигналов) является поточечным произведением их преобразований Фурье. Но редко применяется ввиду небольшого размера ядер свертки
- Обычно применяется подход im2col: фрагменты или патчи входного изображения разворачиваются в матрицу, где каждая строка соответствует векторизованному фрагменту. После чего эта матрица умножается на вытянутое ядро свертки [4]
- Последний используется как метод по умолчанию в большинстве фреймворков глубокого обучения

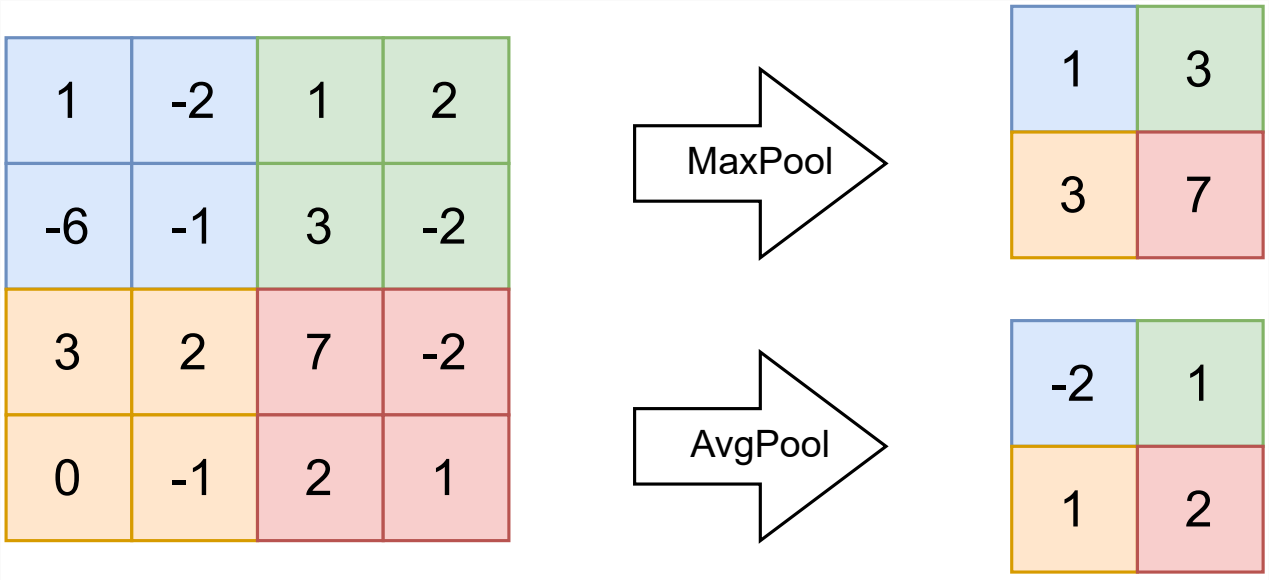
Max/Avg pooling

- Мы уже выяснили, что за счет контроля типа сверток, страйда и паддинга, размерность карт признаков может как увеличиваться, так и уменьшаться
- Однако для задач со входом в высоком разрешении потребуется очень большое количество сверточных слоев (и параметров в них), чтобы уменьшить разрешение и увеличить reseptive field
- Предлагаемое решение по сути своей работает аналогичными сверткам образом: скользящим окном поканально проходимся по изображению/картам признаков, но вместо линейной комбинации, как в свертках, применяем усреднение или взятие максимума

Max/Avg pooling II

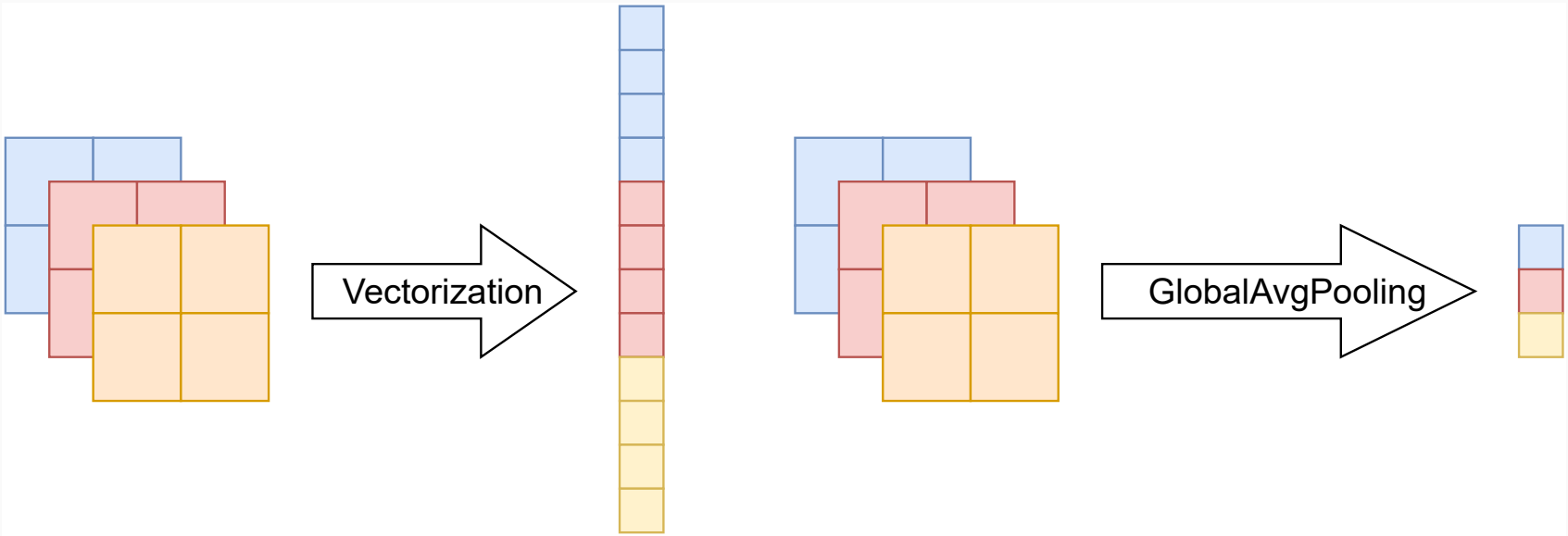
- Обычно как размер ядра пулинг слоя, так как и величина страйда равны двум
- Таким образом, получили быстрый способ уменьшать разрешение карт признаков. В общем случае имеем

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1$$



Global Average Pooling [5]

- Отдельно стоит отметить специальный пулинг слой для задач классификации – global average pooling
- Вместо того, чтобы векторизовывать финальный тензор с картами признаков простым вытягиванием всех значений в вектор (а затем применить полносвязную сеть), предлагается усреднить значения отдельно по каждой карте признаков



Global Average Pooling II

- Нужно это по очень простой причине: при обычной векторизации мы должны четко понимать каково итоговое разрешение карт признаков, чтобы выделить под полносвязные слои фиксированное количество параметров
- Если разрешение изображения изменить, изменится и размерность вектора на вход полносвязной сети, что недопустимо
- Вариант с global average pooling'ом решает эту проблему – размерность входного вектора всегда будет равна числу каналов/карт признаков после последнего блока сверточной части сети

Проблемы глубоких сверточных сетей

- Еще в двухтысячных было показано, что добавление все большего и большего числа сверточных слоев не обязательно приводит к улучшению результатов в задачах компьютерного зрения
- Наоборот, качество может заметно падать. И рассмотренные ранее подходы к инициализации параметров проблему полностью не решают

Residual connections

- Предлагается следующая идея: вместо того, чтобы строить архитектуру сети в виде последовательного применения блоков из некоторого линейного преобразования и функции активации (по сути, последовательность вложенных друг в друга функций), предлагается добавить остаточные пути (residual connections), "огибающие" блоки

Residual connections II

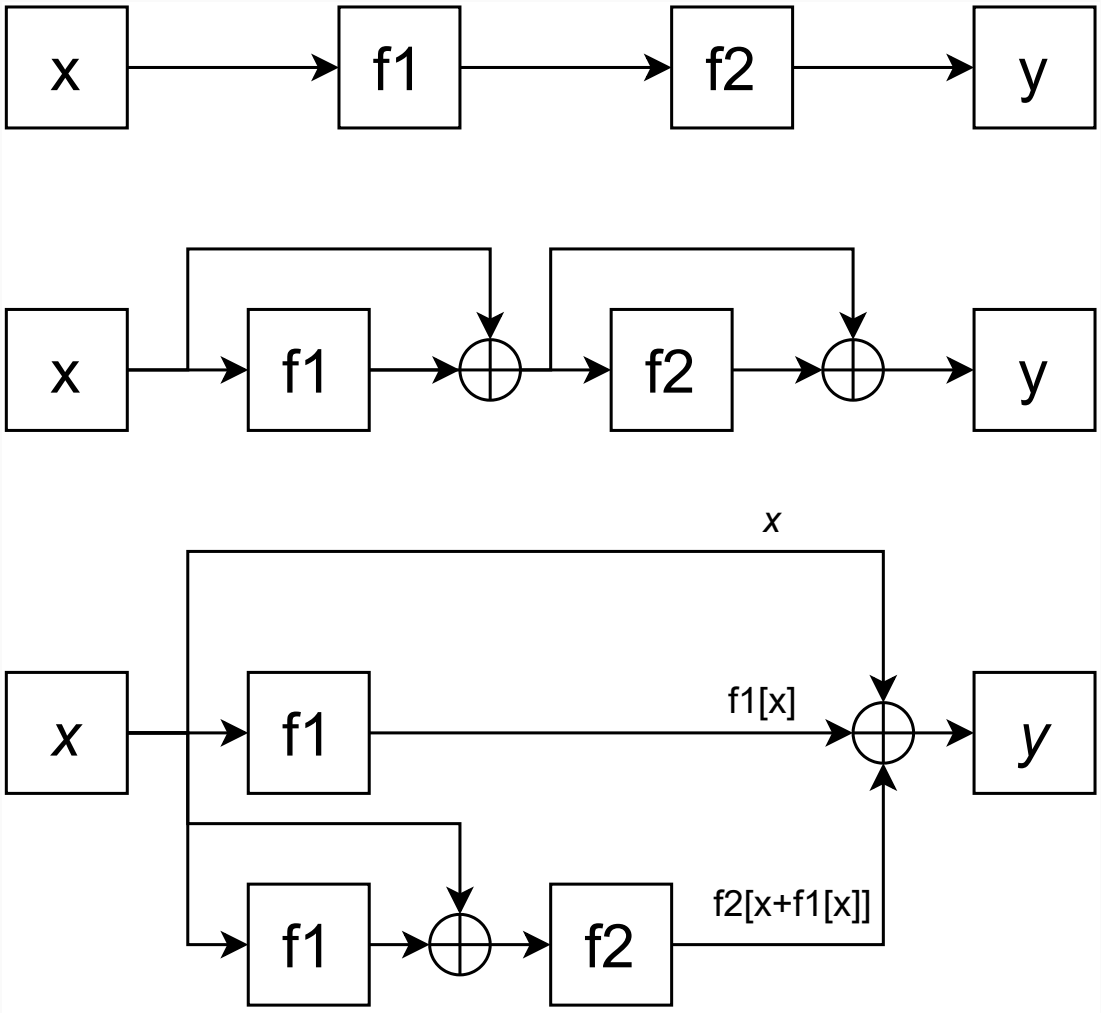


Рис. 2: Визуализация residual connections [6]

Residual connections III

- Теперь в вычислительном графе всегда будет существовать пути, которые будут позволять даже самым первым слоям сети вносить прямой вклад в ее финальный эмбеддинг
- Есть и недостатки – проблема взрывающихся градиентов: дисперсия в местах соединения путей увеличивается вдвое (предполагая использование подходящей), что в целом приводит к ее экспоненциальному росту (сценарий использования функции активации ReLU и инициализации Kaiming)
- Простейшее решение – шкалировать сумму на выходе residual-блока на $\frac{1}{\sqrt{2}}$
- Но есть и более общие варианты решения данной проблемы – нормализация

Internal covariance shift

- В рамках обучения нейронных сетей (не умаляя общности, пусть речь идет о полносвязных слоях) мы часто сталкиваемся с проблемой: изменение параметров текущего линейного слоя приводит к смещению распределения активаций, которые пойдут на вход следующему линейному слою, тем самым раз за разом процесс обучения усложняется
- Контроль над моментами позволил бы упростить и ускорить этот процесс

Нормализация в глубоком обучении

- В предыдущем семестре на курсе по машинному обучению мы разобрали важность нормализации признаков, узнали, зачем это нужно (или не нужно?) делать и познакомились с набором способов это проделать
- В глубоком обучении список методов существенно расширяется
- Связано это как с тем фактом, что нормализацию приходится производить не только на входном слое, но и на всех скрытых, а также с тем фактом, что обучение происходит итеративно небольшими партиями (батчами)

Batch Normalization

- Среди этих методов батч-нормализация (BatchNorm, BN) является одним из самых популярных и работает (в роли отдельного слоя) следующим образом.
- Сначала для каждой компоненты входа вычисляется среднее и (смещенное) стандартное отклонение по всему батчу
- Затем с их помощью производится покомпонентная нормализация входа
- Наконец, на третьем шаге над нормализованными активациями производится афинное преобразование, параметры которого также являются обучаемыми
- Вопрос: зачем нужен последний шаг? Наводящий вопрос: а что будет, если после этого BN-слоя идет слой сигмоид?

Batch Normalization II

- Наличие этих двух параметров (для каждой компоненты входа) в теории позволяет выучить среднее и СКО и по сути превратить BN-слой в identity-слой – если это было бы оптимальным способом уменьшения значения лосс-функции
- BN-слой предоставляет контроль над средними и СКО входа и тем самым упрощает обучение сети
- Формально

$$y = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i, \quad \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2$$

- Инициализируют γ и β 1 и 0 соответственно

Batch Normalization III

- Так, подождите, начинали этот блок с проблемы internal covariance shift'a. Каково влияние BatchNorm'a? Все просто, распределение выходов теперь зависит не от всех параметров в линейном слое, а от набора γ и β
- Вопрос: как вы думаете, что происходит с BN-слоями при эксплуатации (инференсе) сети?

Batch Normalization IV

- Что до сверточных сетей, то батч-нормализация производится целиком для каждой карты признаков (мы ведь хотим сохранить свойство инвариантности обработки разных частей на карте признаков), т.е. параметры γ и β общие
- Эмпирически было показано, что покомпонентная батч-нормализация сглаживают ландшафт лосс-функции, что позволяет использовать большую скорость обучения
- Без проблем тоже не обошлось: маленький размер батча, зависимость значений лосс-функции для наблюдения от того, кто вместе с ним попал в батч, вопросы дообучения сетей с BatchNorm под свою задачу.
- Больше про мотивацию, производные для backprop'а и другие детали см. в оригинальной статье [7]

Примеры нормализаций

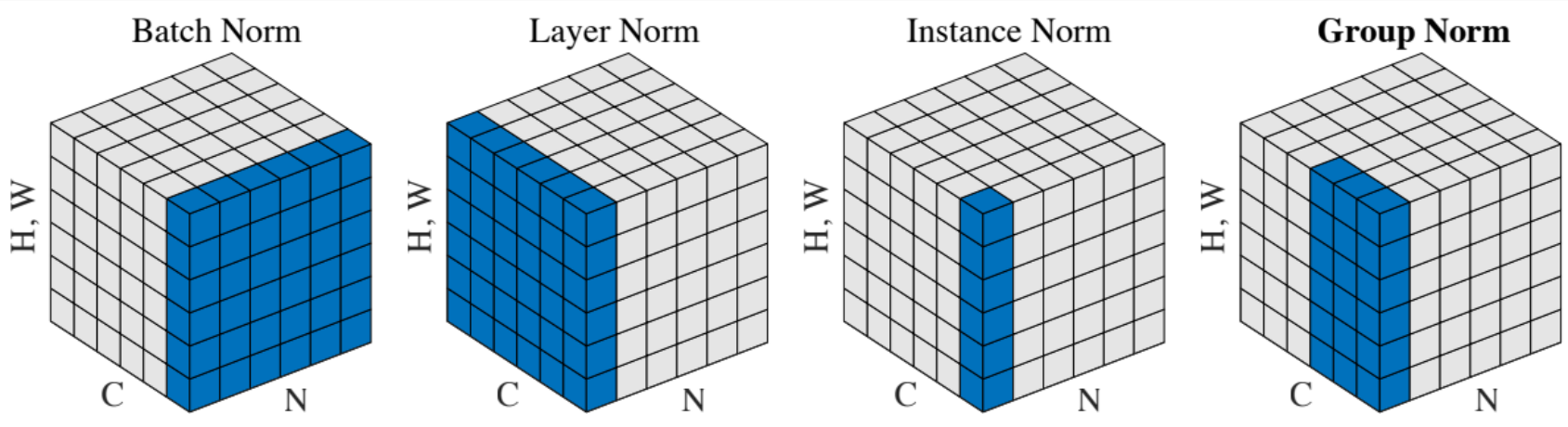
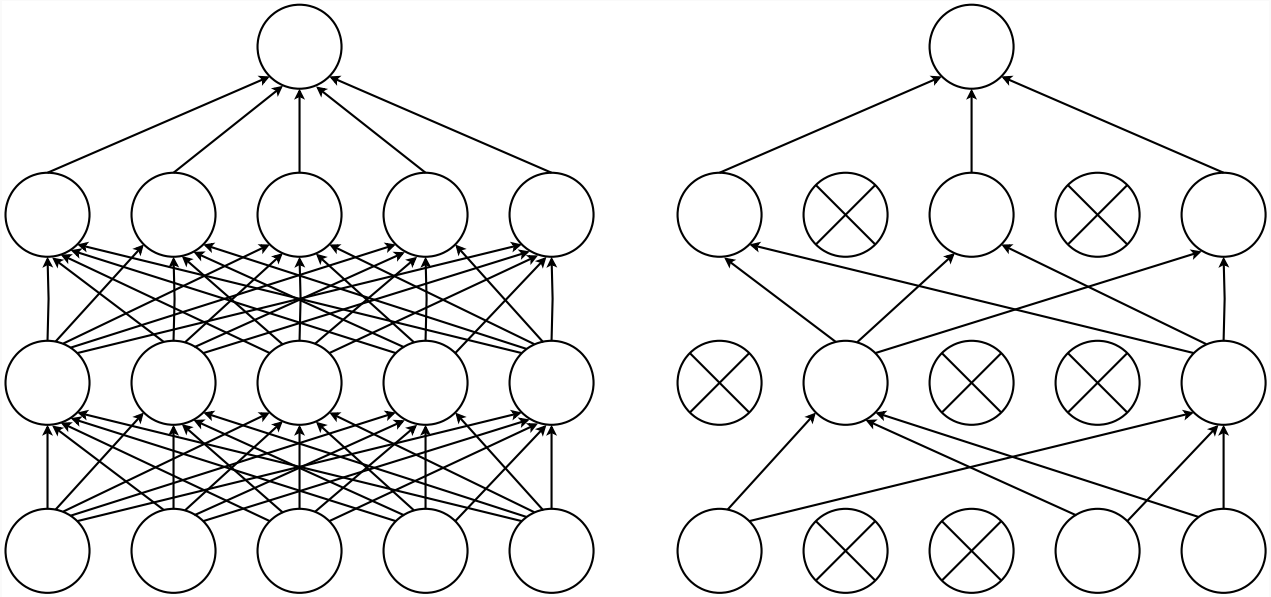


Рис. 3: Различные виды нормализаций нейронных сетей [8]

Dropout [9]

- Ввиду того, что среднее и СКО оцениваются по батчу, BatchNorm выступает в роли регуляризатора: наблюдения зашумляются
- Вот еще одна идея для регуляризации сети – dropout
- Предлагается независимо "отключать" нейроны на скрытом слое с некоторой заранее заданной вероятностью p



Dropout: тренировка и инференс II

- Во время тренировки выключенные нейроны передают нули по сети, тогда как в рамках теста/инференса все нейроны активны, но домножаются на $1 - p$
- Действительно, в рамках тренировки мы ожидаем увидеть значение активации равное $(1 - p)x + p0 = (1 - p)x$. В рамках инференса мы должны это скомпенсировать
- Есть и альтернатива: можно в рамках тренировки производить эту компенсацию сразу (деление всех незануленных выходов на $1 - p$). Называется Inverted Dropout и зачастую именно так дропаут реализован в фреймворках глубокого обучения. Есть идеи, почему так?
- Часто вместе с дропаутом упоминают ансамбль нейронных "под-сетей". Как вы думаете, почему?

Dropout: замечания

- В рамках алгоритма обратного распространения ошибок ничего не меняется, просто учитывается то, какие из нейронов были активны
- Не принято использовать различные значения p между слоями (не относится к ситуациям, когда ваша сеть представляет собой композицию некоторого числа самостоятельных сетей-блоков)
- Обычно p, как и другие гиперпараметры, подбирают по результатам на отложенной выборке. Чаще всего выбирают из 0.1 и 0.5

Итого

- Разобрались с основными составляющими сверточных нейронных сетей: свертками и слоями пулинга
- Познакомились с еще одним способом борьбы с проблемой затухающих градиентов – residual connections
- Узнали про способы нормализации и регуляризации в нейросетевых архитектурах: batch-нормализация и dropout
- На следующем занятии отработаем изученное на практике, а именно посмотрим на использование сверточных сетей в основных задачах компьютерного зрения

Использованные источники i

1. *Albumentations-team*. **Image augmentations pipeline.** URL: https://albumentations.ai/docs/getting_started/image_augmentation/.
2. *Dumoulin V.*, *Visin F.* **A guide to convolution arithmetic for deep learning.** // arXiv e-prints. 2016. DOI: 10.48550/arXiv.1603.07285. arXiv: 1603.07285.
3. *Araujo A.*, *Norris W.*, *Sim J.* **Computing Receptive Fields of Convolutional Neural Networks.** // Distill. 2019. DOI: 10.23915/distill.00021. https://distill.pub/2019/computing-receptive-fields.
4. **How convolution layer is implemented in PyTorch.** URL: https://discuss.pytorch.org/t/how-to-implement-a-convolutional-layer/68211.

Использованные источники ii

5. *Lin M.*, *Chen Q.*, *Yan S.* **Network In Network.** // arXiv e-prints. 2013. DOI: 10.48550/arXiv.1312.4400. arXiv: 1312.4400.
6. *Prince S. J.* **Understanding Deep Learning.** The MIT Press, 2023. URL: http://udlbook.com.
7. *Ioffe S.*, *Szegedy C.* **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.** // arXiv e-prints. 2015. DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167.
8. *Wu Y.*, *He K.* **Group Normalization.** // arXiv e-prints. 2018. Март. DOI: 10.48550/arXiv.1803.08494. arXiv: 1803.08494 [cs.CV].
9. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting.** /. N. Srivastava [и др.] // Journal of Machine Learning Research. 2014. Т. 15, № 56. С. 1929—1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.