Learning Strides in Convolution Neural Network

Created time	@January 30, 2023 11:40 PM
Tags	CNN CV ML
Telegram	https://t.me/before_relu_or_after
Автор	Савченко Я.
Редакторы	Гончаренко А., Щербин А.
Код	https://github.com/google-research/diffstride
Статья	https://arxiv.org/pdf/2202.01653.pdf

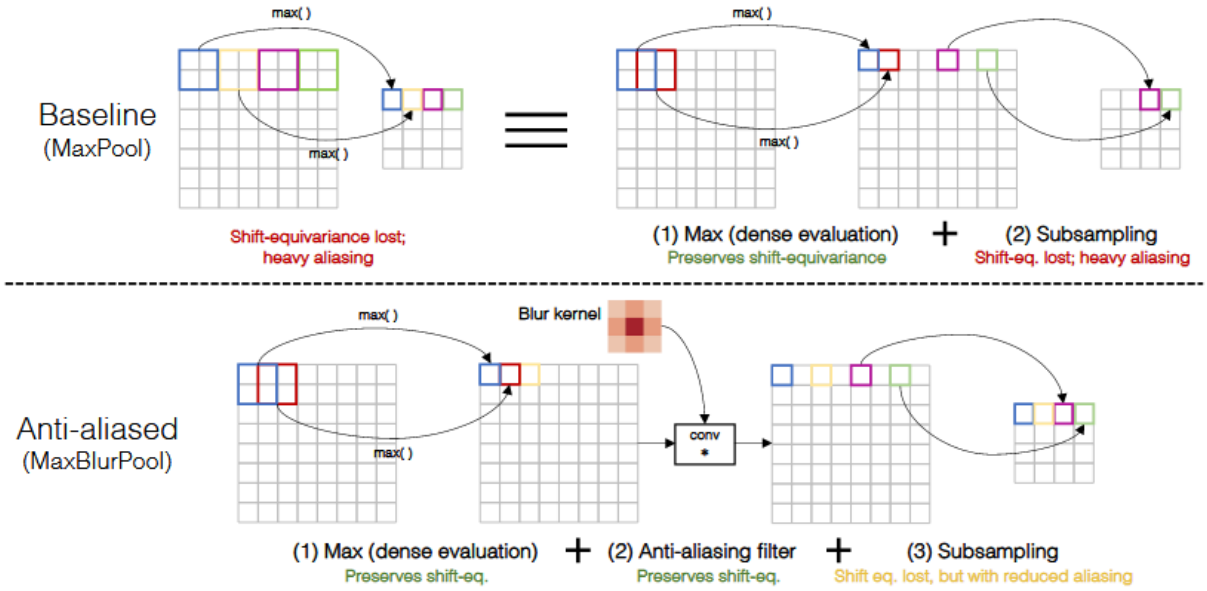
Сверточные нейронные сети обычно содержат операторы, понижающие пространственную размерность тензоров, такие как свертки или пулинги. Важным гиперпараметром этих операторов является страйд. Поскольку страйды недифференцируемые, то для подбора лучшей конфигурации страйдов используется кросс-валидация либо NAS. Сложность этого подхода заключается в том, что пространство поиска растет экспоненциально с числом слоев, понижающих размерность. В этом обзоре мы расскажем про

статью, авторы которой предложили новый оператор DiffStride, понижающий размерность, с обучаемыми вещественными страйдами. Благодаря обучаемости страйдов, их подбор с обычным градиентным спуском требует меньших вычислительных мощностей.

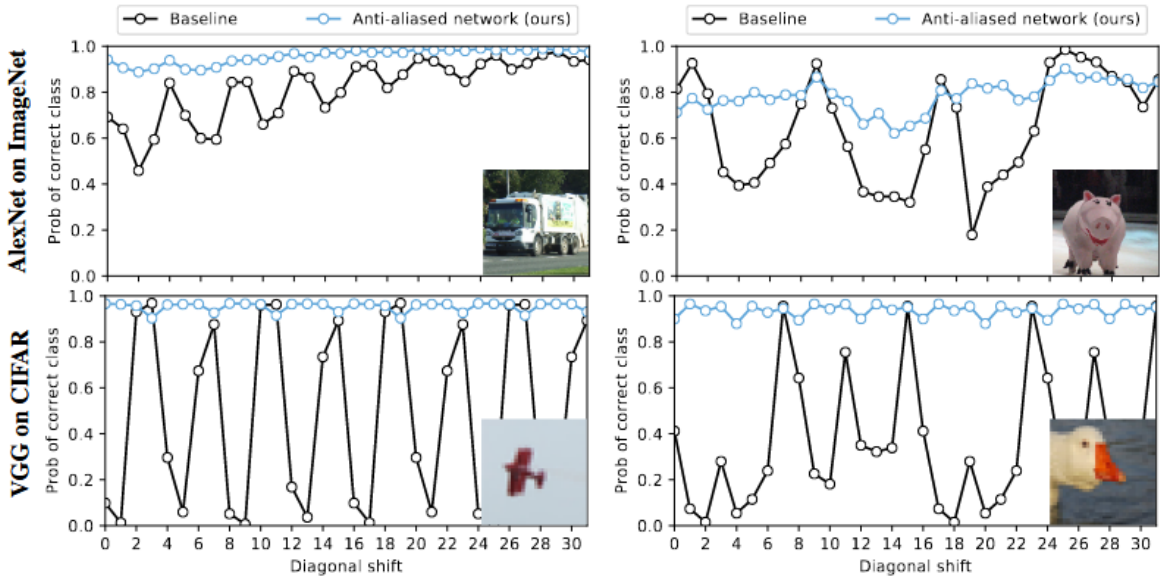
Введение

Чем глубже в сети находится сверточный слой, тем более высокоуровневые признаки он позволяет извлечь. Из неупорядоченной информации (тензора с большим размером и скажем, тремя каналами) мы получаем упорядоченную (тензор с большим числом каналов и маленьким размером). Использование страйдов позволяет уменьшать размер промежуточных тензоров и сокращать число операций с плавающей точкой. К тому же, используя страйды, мы влияем на размер рецептивного поля (*receptive field*). В общем, страйды - это важная вещь, без них не обойтись.

До выхода обсуждаемой нами статьи другие исследователи тоже пытались как-то усовершенствовать свертки со страйдами и пулинги. Например, *Richard Zhang* писал о том, что оператор пулинга со страйдом (или свертки, для нее тоже это справедливо) можно разделить на две части: пулинг (или свертка) со страйдом единица и подвыборка (*sub-sampling*) с шагом, равным страйду. Если первая операция устойчива по отношению к сдвигу, то вторая - нет. И он предложил добавить дополнительную операцию перед подвыборкой (а именно - свертку с гауссовым фильтром) и объяснил, почему это повысит устойчивость к сдвигу.



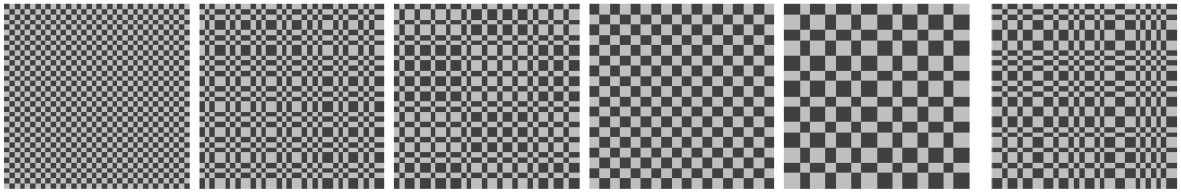
Сверху - обычный MaxPool, снизу - модификация, более устойчивая к сдвигу. Картинка из статьи Richard Zhang, Making Convolutional Networks Shift-Invariant Again.



Вероятность выбрать верный класс при классификации изображения в зависимости от сдвига. Сдвиг производится вдоль диагонали изображения. Видно, что после замены обычных сверток со страйдами на более хитрый оператор метрика стала более устойчивой. Картинка из статьи Richard Zhang, Making Convolutional Networks Shift-Invariant Again.

Benjamin Graham придумал пулинг с дробными страйдами. Мотивация его была следующая: использование даже страйдов 2x2 уже приводит к тому,

что 75% информации из исходного тензора теряется. И ему захотелось изобрести какие-то другие страйды (не единичные), с которыми информация в сети теряется не так быстро. Величина страйда, когда он нецелый - это просто отношение сторон до и после пулинга со страйдом. А области, по которым нужно пулить, можно генерировать случайным образом.



Первое изображение - сетка 36x36. Следующие за ним - непересекающиеся области (обозначены разным цветом) для пулинга со страйдами 32, 2, 2, 5 (используются псевдослучайные последовательности) и 2 (случайные последовательности). Картинка из статьи Benjamin Graham, Fractional Max-Pooling.

Левое верхнее изображение имеет разрешение 384x256. Остальные изображения - это выход после шести слоев AvgPool со страйдом 2 со случайными непересекающимися регионами. Эти пять изображений увеличены, в реальности сторона каждого изображения после пулингов уменьшилась в восемь раз. Картинка из статьи Benjamin Graham, Fractional Max-Pooling.

Если подбирать страйды для конкретной сети с помощью, например, кросс-валидации или NAS, то оказывается, что число возможных конфигураций параметров экспоненциально растет с числом слоев, понижающих

размерность. И хотя нецелые страйды делают архитектуру нейросети более гибкой, они только увеличивают размер пространства поиска и делают задачу их выбора сложнее. Вот было бы здорово, если бы страйды можно было бы обучать при помощи градиентного спуска, как обычные веса, не правда ли?

SpectralPooling

В 2015 году в статье *Oren Rippel et al, Spectral Representations for Convolutional Neural Networks* было предложено использовать дискретное преобразование Фурье (ДПФ) для уменьшения пространственного разрешения тензоров. Напомним, ДПФ имеет следующий вид:

$$\mathcal{F}(x)_{mn} = \frac{1}{\sqrt{HW}} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} x_{hw} e^{-2\pi i \left(\frac{mh}{H} + \frac{nw}{W}\right)}, \forall m \in \{0, \dots, H-1\}, \forall n \in \{0, \dots, W-1\}. \tag{1}$$

Пусть x - это исходный сигнал из $\mathbb{R}^{H \times W}$ (то есть действительный), а y - это его Фурье-образ из $\mathbb{C}^{H \times W}$ (уже комплексный). ДПФ является линейным, и обратное к нему преобразование является к сопряженным к нему: $F(.)^{-1} = F(.)^*$

Когда x действительный, как в нашем случае, то Фурье-преобразование от него является эрмитовым (то есть транспонированная матрица равна комплексно сопряженной). Для нас это значит, что, воспользовавшись эрмитовостью, мы можем реконструировать x только по положительной половине частот.

ДПФ является дифференцируемым, и его производная равна обратному ДПФ. Таким образом, если мы захотим посчитать градиент лосса по y, то мы всегда можем выразить его через производную по x:

$$x \in \mathbb{R}^{H \times W}, y = \mathcal{F}(x), \frac{\partial \mathcal{L}}{\partial x} = \mathcal{F}^*\left(\frac{\partial L}{\partial y}\right) = \mathcal{F}^{-1}\left(\frac{\partial L}{\partial y}\right).$$

А теперь перейдем непосредственно к SpectralPooling. Для изображений и аудиозаписей отношение сигнала к шуму неравномерно распределено по частотам. Большая часть сигнала соответствует низким частотам, в то время как высокие частоты обычно кодируют шум. Благодаря этому свойству, мы можем использовать преобразование Фурье для фильтрации. Например, обрабатывая изображение, при помощи прямого ДПФ можно обрезать область высоких частот, сохранив значительную часть информации. Делая обратное ДПФ, мы получаем ту же картинку, только более размытую и уменьшенную в размере:

Результат применения MaxPooling и SpectralPooling на одном и том же изображении. Нижний ряд означает долю частот, оставшихся после кропа в Фурье-простанстве. После SpectralPooling картинка была заресайжена к исходному размеру. Изображение из статьи Oren Rippel et al, Spectral Representations for Convolutional Neural Networks

Пусть вход - это сигнал x из $\mathbb{R}^{H \times W}$, $S = (S_h, S_w)$ - это страйды. Тогда работу SpectralPooling можно описать следующими пунктами:

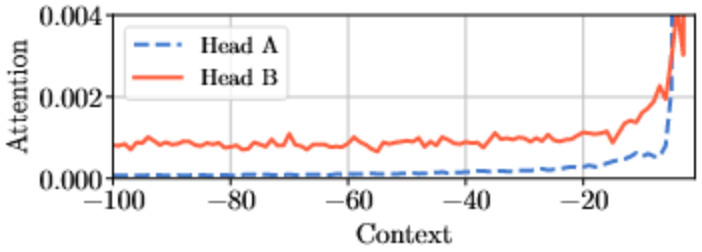
1. Считаем ДПФ $y = F(x)$. Для простоты считаем, что нулевой частоте соответствует центр матрицы y
2. Берем bounding box с размерами $[H/S_h] \times [W/S_w]$ и кропаем вокруг центра, получаем y' из $\mathbb{C}^{[H/S_h] \times [W/S_w]}$.
3. Получаем x' из $\mathbb{R}^{[H/S_h] \times [W/S_w]}$ при помощи обратного ДПФ $x' = F^{-1}(y')$

С помощью SpectralPooling можно уменьшать тензоры не только в целое число раз: в SpectralPool страйд не обязательно должен быть целым - главное, чтобы результирующие размеры картинки получались целочисленными Еще одно преимущество SpectralPooling состоит в том, что урезание высоких частот повышает устойчивость к сдвигу. SpectralPooling является дифференцируемым по отношению ко входам, но не по отношению к своим страйдам, что все еще не позволяет делать их обучаемыми.

DiffStride

Предложенный в статье подход имеет много общего со SpectralPooling. Так же, как и в SpectralPooling, при использовании этого оператора происходит кроп тензора в Фурье-пространстве. Различие в том, что *bounding box* не фиксированный - DiffStride выучивает размеры бокса при помощи бекпропа. Этот *bounding box* W параметризован размерами входа, сглаживающим фактором R и страйдами. W получается как внешнее произведение двух одномерных маскирующих функций для двух координат - x и y. На создание этих маскирующих функций авторов вдохновила NLP-шная статья про внимание от исследователей из Facebook AI Research - *Adaptive Attention Span in Transformers*. Давайте подробнее разберем, в чем заключалась их идея.

Авторы статьи обратили внимание на то, что разные головы трансформеров в слое *self-attention* нуждаются в разном количестве контекста. Это проиллюстрировано на картинке ниже:



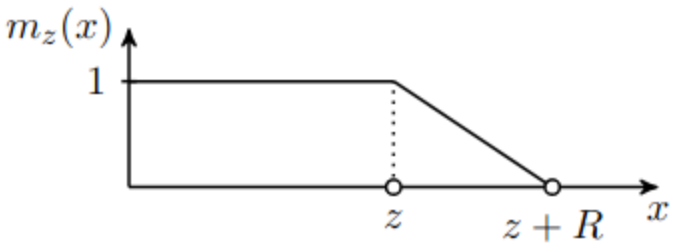
Внимание для двух голов трансформера A и B. Голова A в основном учитывает только недавний контекст, а голова B учитывает весь контекст. Картинка из статьи S. Sukhbaatar et al, Adaptive Attention Span in Transformers

Пусть у нас есть токен t. Вес для внимания между этим токеном и ключом r считается по следующей формуле:

$$a_{tr} = \frac{\exp\left(s_{tr}\right)}{\sum_{q=t-S}^{t-1} \exp\left(s_{tq}\right)},$$

где суммирование происходит по всему контекстному окну токена t (в формуле размер контекстного окна равен S). Было предложено для каждой головы трансформера завести маскирующую функцию, которая бы контролировала количество контекста. Эта функция $m_z(x)$ параметризуется числом $z \in [0, S]$, которое сетке нужно выучить.

Вот она:



А вот так выглядит формула:

$$m_z(x) = \min\left[\max\left[\frac{1}{R}\left(R + z - x\right), 0\right], 1\right],$$

По сути, функция $m_z(x)$ нужна только для того, чтобы начиная с z-го токена голова трансформера могла бы потихоньку “забывать” более старый контекст, и соответствующие ему экспоненты в софтмаксе суммировать с коэффициентами меньше единицы. R в формуле - это величина размытия, чтобы не обрубать окно контекста слишком резко. С учетом модификаций веса для влияния выглядят так:

$$a_{tr} = \frac{m_z(t-r)\exp(s_{tr})}{\sum_{q=t-S}^{t-1} m_z(t-q)\exp(s_{tq})}.$$

Данное выражение дифференцируемо по z. Для каждой головы свой параметр z выучивается с помощью обратного распространения ошибки. В оригинальной статье еще добавляли в функцию потерь член с l_1-регуляризацией на z. Ограничение контекста, как описано выше, приводит к уменьшению потребления памяти и вычислительных ресурсов: если удается ограничить контекст, значит, надо меньше считать.

А теперь вернемся к маскам для DiffStride, которые были вдохновлены масками для механизма внимания. Напомню тем, кто успел забыть: основная идея DiffStride заключается в том, чтобы сделать обучаемый *bounding box* W для кропа в Фурье-пространстве. А бокс этот будет равен внешнему произведению от двух обучаемых масок:

$$\text{mask}^{h}_{(S_h,H,R)}(m) = \min\left[\max\left[\frac{1}{R}(R + \frac{H}{2S_h} - |\frac{H}{2} - m|), 0\right], 1\right], m \in [0, H] \quad (3)$$

$$\text{mask}^{w}_{(S_w,W,R)}(n) = \min\left[\max\left[\frac{1}{R}(R + \frac{W}{2S_w} + 1 - n), 0\right], 1\right], n \in [0, \frac{W}{2} + 1] \quad (4)$$

где S_h, S_w - это страйды. Благодаря эрмитовости ДПФ, мы рассматриваем только положительные частоты вдоль горизонтальной оси, в то время как вертикальную маску мы отражаем относительно нулевой частоты. Поэтому формулы немного разные.

Маска W используется в двумя способами:

- W применяется к Фурье-образу при помощи поэлементного умножения
- Мы кропаем Фурье-коэффициенты там, где маска W равна нулю

Первая из этих двух операций дифференцируемая, а вторая - нет. Поэтому перед применением кропа используется операция *stop gradient*

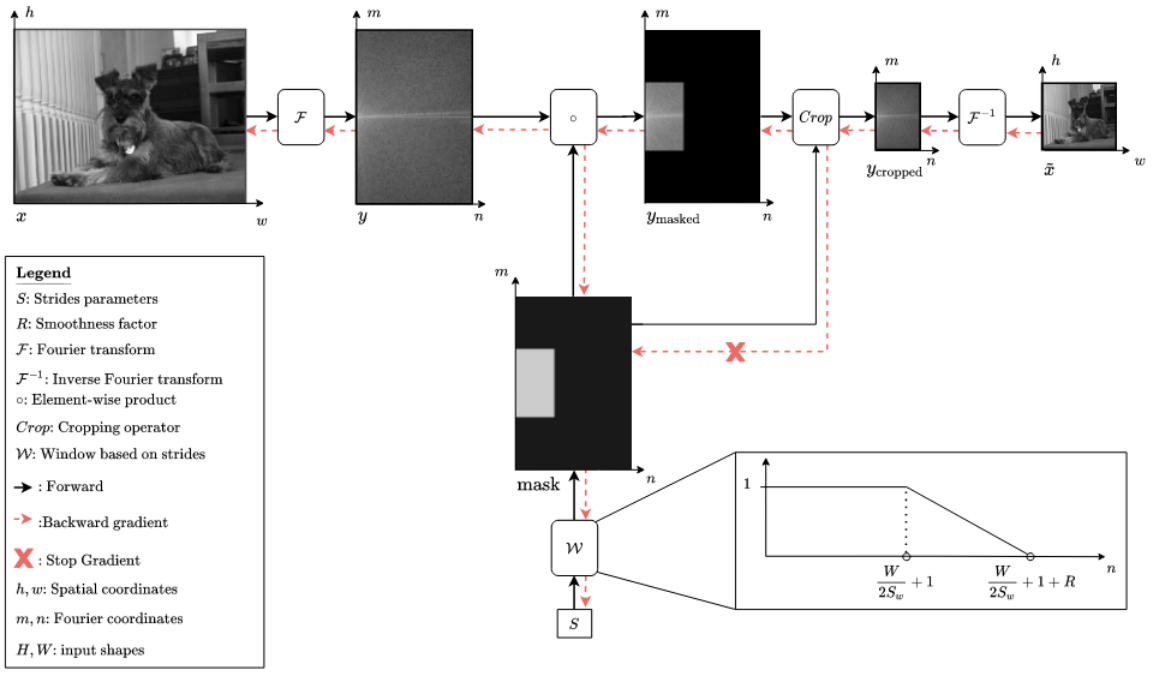


Схема прямого и обратного прохода DiffStride для одноканального изображения. Картинка из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network

В обычных свертках или в MaxPool обычно страйды для двух измерений совпадают, тут же, как показал эксперимент, разные страйды для h и w подходят лучше. Размытие R - это единый параметр для всех слоев, потому что делать эту переменную гиперпараметром противоречит идее избавления от страйдов как от гиперпараметра.

А еще из-за дифференцируемости страйдов можно добавить к лоссу регуляризацию определенного вида, которая будет оптимизировать число операций и память модели. Действительно, сложность 2D-свертки пропорциональна произведению $H^L \times W^L$, где L - это индекс слоя. H^L и W^L выражаются через размеры предыдущего слоя следующим образом:

$$H^L \times W^L = [\frac{H^{L-1}}{S_h^{L-1}} + 2 \times R] \times [\frac{W^{L-1}}{S_w^{L-1}} + 2 \times R]$$

А теперь пренебрежем всеми слагаемыми с $2 \times R$ и запишем суммарную сложность всех сверток. Она будет пропорциональна сумме из вот таких произведений:

$$\sum_{l=1}^{l=L} \prod_{i=1}^{l} \frac{1}{S_h^i \times S_w^i}$$

На основании этого мы можем добавить вот такую регуляризацию к лоссу, чтобы оптимизировать вычислительную сложность модели:

$$\lambda J((S^l)_{l=1}^{l=L}) = \lambda \sum_{l=1}^{l=L} \prod_{i=1}^{l} \frac{1}{S_h^i \times S_w^i},$$

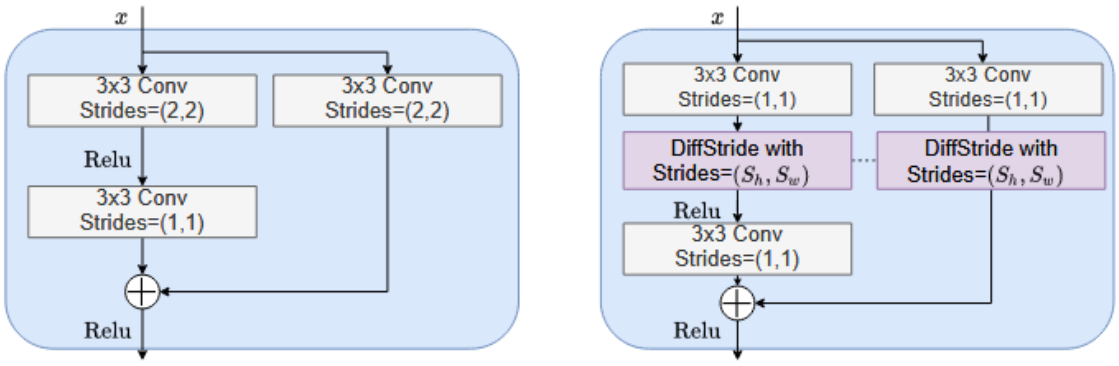
где λ - это регуляризационный коэффициент. В следующей части обзора мы увидим, что выбор λ позволяет “обменять” точность на вычислительную эффективность сети.

Эксперименты

В отличие от сетей, в которых выход l-го слоя идет на вход $l+1$-го, ResNet устроен не так. В ResNet’ах есть два типа блоков:

- *identity blocks*, сохраняющие разрешения тензора после выхода из такого блока
- *shotcut blocks*, понижающие пространственный размер тензора и увеличивающий число каналов в тензоре (изображен ниже)

Для своих экспериментов авторы статей заменили свертки со страйдами в *shotcut*-блоках на свертку со страйдом 1 и последующий DiffStride.



(a) Residual block with a strided convolution.

(b) Residual block with a shared DiffStride layer.

Слева - обычный shortcut-блок, справа - shortcut-блок с DiffStride. Картинка из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network

Авторы статьи протестили DiffStride на восьми классификационных задачах. Единственное отличие в архитектуре состояло в замене сверток со страйдами на свертку с единичным страйдом и DiffStride. Чтобы увидеть

разницу в метриках именно за счет обучаемости страйдов, а не кропов в Фурье-пространстве, авторы также сравниваились с SpectralPool.

Классификация аудио

Авторы включили следующие задачи в эксперименты:

- Классификация акустических сцен (н., звуки шума толпы, метро и т.д..)
- Обнаружение пения птиц
- Музыкально-инструментальная классификация
- Классификация речевых команд

Модели с DiffStride демонстрировали такую же или лучшую метрику, как и модели с обычными свертками или SpectralPool:

Setting	Single-task			Multi-task		
Task	Strided Conv.	Spectral	DiffStride	Strided Conv.	Spectral	DiffStride
Acoustic scenes	**99.1** $\pm$ 0.2	98.6 $\pm$ 0.1	98.6 $\pm$ 0.2	**97.7** $\pm$ 0.4	**97.7** $\pm$ 0.7	**97.7** $\pm$ 0.3
Birdsong detection	78.8 $\pm$ 0.3	79.7 $\pm$ 0.3	**81.3** $\pm$ 0.1	77.3 $\pm$ 0.2	77.8 $\pm$ 0.3	**78.6** $\pm$ 0.5
Music (instrument)	72.6 $\pm$ 0.3	72.9 $\pm$ 0.5	**75.4** $\pm$ 0.0	69.8 $\pm$ 0.4	70.4 $\pm$ 0.4	**73.0** $\pm$ 0.8
Music (pitch)	91.8 $\pm$ 0.1	90.1 $\pm$ 0.0	**92.2** $\pm$ 0.1	89.4 $\pm$ 0.3	87.6 $\pm$ 0.7	**89.9** $\pm$ 0.3
Speech commands	87.3 $\pm$ 0.1	88.5 $\pm$ 0.3	**90.5** $\pm$ 0.3	83.5 $\pm$ 0.6	83.9 $\pm$ 0.4	**86.2** $\pm$ 0.8
Mean Accuracy	85.0 $\pm$ 9.3	86.0 $\pm$ 9.2	**88.3** $\pm$ 8.7	83.5 $\pm$ 10.0	83.5 $\pm$ 9.6	**85.0** $\pm$ 8.9

Точность на валидации (в %). Single-task - это эксперименты, где для каждой задачи (acoustic scenes, birdsong detection и т.д.) использовалась отдельная сеть. В multi-task для всех задач сеть была одна. Цифры, указанные после ± - это дисперсия по 3 экспериментам. Таблица из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network

Классификация изображений

Авторы использовали архитектуру ResNet-18 как бейзлайн, а сравнивались на датасетах CIFAR100 и Imagenet. Сначала они решили инициализировать часть страйдов в ResNet-18 значениями, которые случайно засемплировали. В таблице внизу видно, что изменения страйдов сильно отражаются на метрике. Тем не менее, моделям с DiffStride удалось, несмотря на неудачную инициализацию, сойтись к хорошим метрикам:

	CIFAR10			CIFAR100		
Init. Strides	Strided Conv.	Spectral	DiffStride	Strided Conv.	Spectral	DiffStride
(2, 2, 2)	91.4 ± 0.2	92.4 ± 0.1	**92.5** ± 0.1	66.8 ± 0.2	**73.7** ± 0.1	73.4 ± 0.5
(2, 2, 3)	90.5 ± 0.1	92.2 ± 0.2	**92.8** ± 0.1	63.4 ± 0.5	**73.7** ± 0.2	73.5 ± 0.0
(1, 3, 1)	90.0 ± 0.4	91.1 ± 0.1	**92.4** ± 0.1	64.9 ± 0.5	70.3 ± 0.3	**73.4** ± 0.2
(3, 1, 3)	85.7 ± 0.1	90.9 ± 0.2	**92.4** ± 0.1	55.3 ± 0.8	69.4 ± 0.4	**73.7** ± 0.4
(3, 1, 2)	86.4 ± 0.1	90.9 ± 0.2	**92.3** ± 0.1	56.2 ± 0.3	69.9 ± 0.2	**73.4** ± 0.3
(3, 2, 3)	82.0 ± 0.6	89.2 ± 0.2	**92.3** ± 0.1	48.2 ± 0.2	66.6 ± 0.5	**73.6** ± 0.4
Mean accuracy	87.7 ± 3.4	91.1 ± 1.1	**92.4** ± 0.2	59.1 ± 6.7	70.6 ± 2.6	**73.5** ± 0.3

Точности на валидации (в %) на CIFAR10 и CIFAR100. Цифры, указанные после ± - это дисперсия по 3 экспериментам.Таблица из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network.

Подобный эксперимент также провели на Imagenet'e:

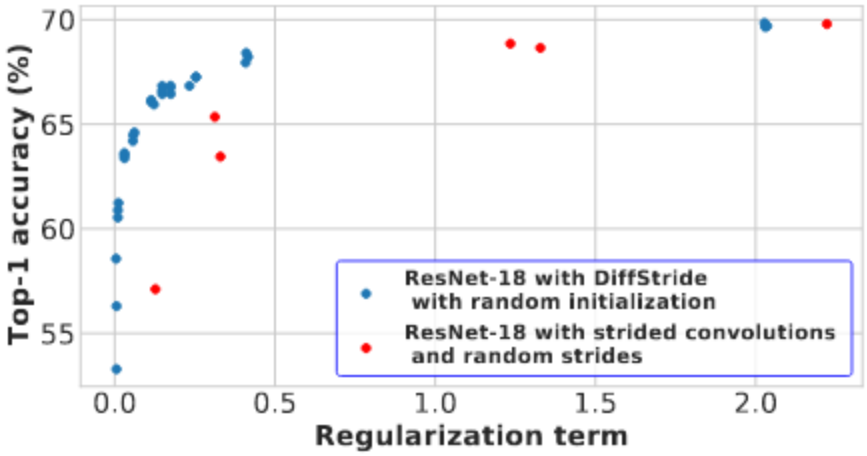
	Top-1			Top-5		
Init. Strides	Strided Conv.	Spectral	DiffStride	Strided Conv.	Spectral	DiffStride
(1, 2, 2, 2)	68.65 ± 0.26	69.01 ± 0.19	**69.66** ± 0.06	88.5 ± 0.15	88.48 ± 0.02	**89.07** ± 0.03
(1, 1, 3, 1)	69.79 ± 0.15	**69.88** ± 0.05	68.22 ± 0.07	**89.43** ± 0.18	89.15 ± 0.07	88.10 ± 0.08
(1, 3, 1, 3)	68.86 ± 0.28	68.63 ± 0.08	**69.41** ± 0.16	88.64 ± 0.15	88.42 ± 0.01	**88.98** ± 0.04
(2, 2, 2, 3)	63.45 ± 0.09	67.16 ± 0.17	**69.53** ± 0.08	85.09 ± 0.04	87.25 ± 0.06	**89.05** ± 0.05
(2, 3, 1, 2)	65.35 ± 0.03	66.35 ± 0.24	**69.42** ± 0.06	86.27 ± 0.05	86.67 ± 0.15	**88.91** ± 0.05
(3, 3, 2, 3)	57.11 ± 0.11	64.44 ± 0.01	**69.43** ± 0.11	80.42 ± 0.11	85.22 ± 0.09	**89.03** ± 0.02
Mean accuracy	65.53 ± 4.49	67.58 ± 1.88	**69.28** ± 0.50	86.39 ± 3.15	87.53 ± 1.36	**88.85** ± 0.35

Точности на валидации (в %) на Imagenet. Цифры, указанные после ± - это дисперсия по 3 экспериментам. SOTA-метрика на Imagenet составляет 90.88%. Таблица из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network

Из результатов для CIFAR100 наглядно следует, что страйды действительно являются критически важными гиперпараметрами для ResNet-18 (точность варьируется от 66.8% до 48.2% между лучшей и худшей конфигурацией). Из всех трех вариантов сверток со страйдами DiffStride показал себя наилучшим образом, продемонстрировав большую устойчивость к начальной инициализации параметров.

Что насчет регуляризации? Авторы провели эксперименты, в которых тренировали ResNet-18 на Imagenet'e, изменяя коэффициент регуляризации λ от 0.1 до 10, всегда инициализируя страйды одним и тем же значением. На графике снизу построена точность в зависимости от вычислительной

эффективности (слагаемое с регуляризацией в лоссе на момент сходимости). Видно, что сети с обычными страйдами и DiffStride при одинаковой метрике имеют разную вычислительную сложность.



Точность ResNet-18 на Imagenet'е в зависимости от вычислительной эффективности. Картинка из статьи Rachid Riad et al, Learning Strides in Convolution Neural Network

Вывод

DiffStride - слои с обучаемыми страйдами - это интересная альтернатива обычным сверткам со страйдами. Однако они не являются панацеей. У них тоже есть недостатки - например, они не работают на TPU. Для TPU нужен статичный вычислительный граф, а во время обучения DiffStride меняет пространственные размеры промежуточных тензоров.