

現場で使えるディープラーニング基礎講座

DAY5

SkillUP AI

前回の復習

- 前回は何を学びましたか？

中間発表

中間発表

- 課題の成果を発表して頂きます。
- 1人当たりの時間は、**発表3分+質疑応答2分=5分**とします。
- 発表内容に入るものの
 - 1. 実装の進捗状況
 - 2. モデルの改良点と識別精度の変遷
 - 3. 学習用データでの識別精度とテスト用データでの識別精度の比較
 - 4. Arxiv.orgなどで見つけた論文とそこから得られた知見
 - 5. 今後の取り組み予定
- 事前に、課題をまとめたNotebookを講師にDMで渡してください。
- 発表時は、そのNotebookをプロジェクターに投影するようにします。時間が限られていますので、原則、個人PCでの発表は受け付けないようにしたいと思います。

畳み込みニューラルネットワーク

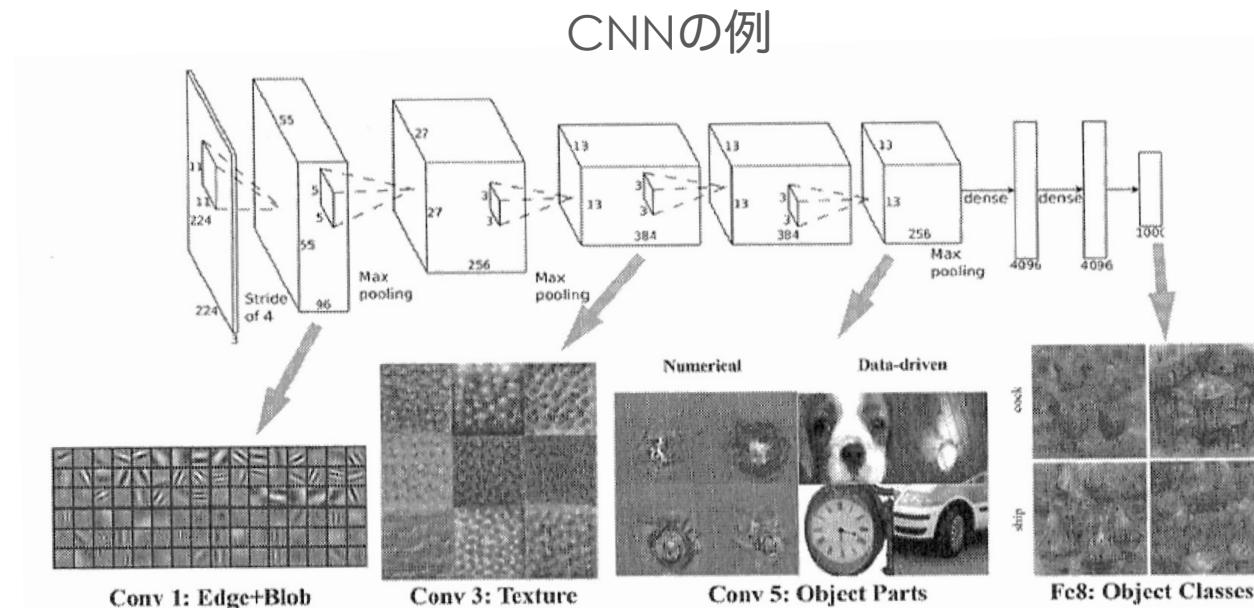
目次

1. CNN概要
2. 置み込み層
3. プーリング層
4. im2col

CNN概要

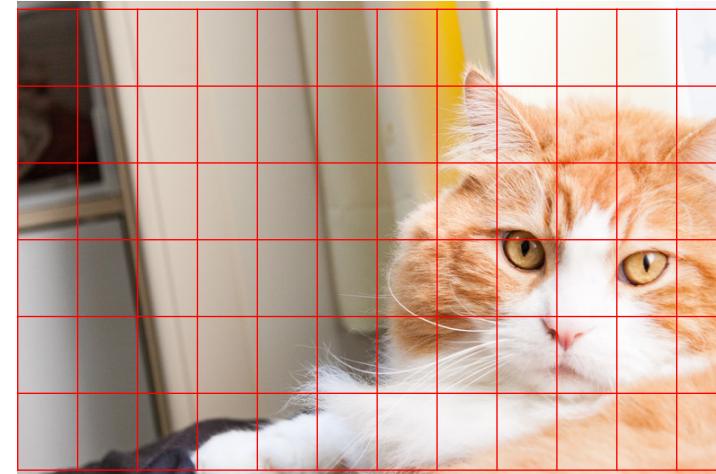
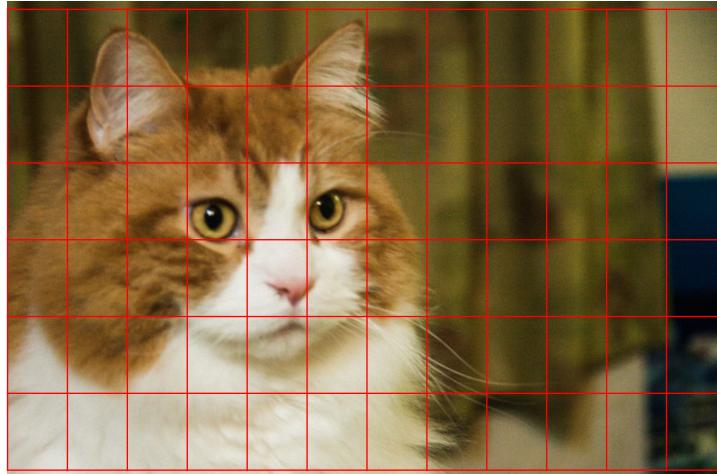
CNN (Convolutional Neural Network) とは

- CNNとは、畳み込み層とプーリング層を用いたニューラルネットワークのこと。
- 主に画像認識に使われる。
- 畳み込み層とプーリング層は、脳の視覚野に関する神経科学の知見をヒントにしている。
参考：『深層学習, 岡谷, p.79』



CNNのコンセプト

- 以下の2枚はどちらも猫の写真。
- でも、顔の位置や向きが異なる。
- どんな猫の写真が入力されてもきちんと猫であると識別できるモデルをつくりたいが、通常のNNでは特徴のズレを考慮できない。
- 特定の写真に依存しない普遍的な特徴を抽出するにはどうすればいいか？
- その難問に応えるべく考案されたのが畠み込み層とプーリング層。



従来の識別手法とCNNの違い



→ 人の考えたアルゴリズム → 答え



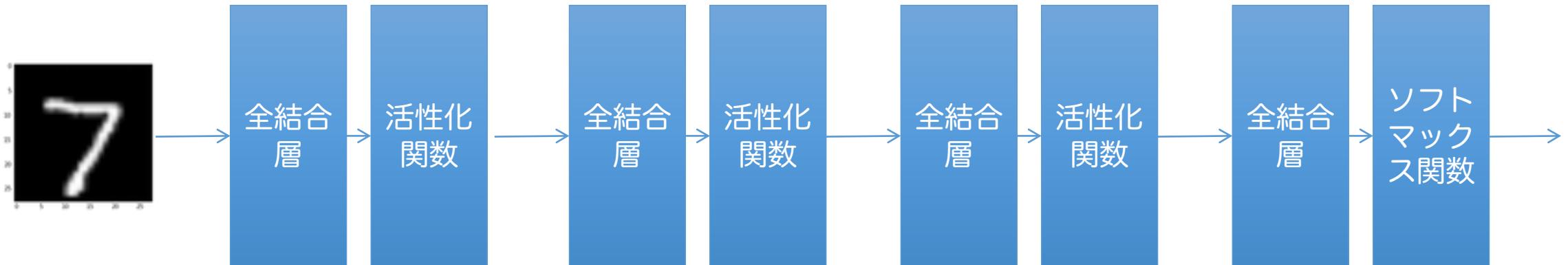
→ 人の考えた特徴量
(SIFT、HOGなど) → 機械学習
(SVM、KNNなど) → 答え



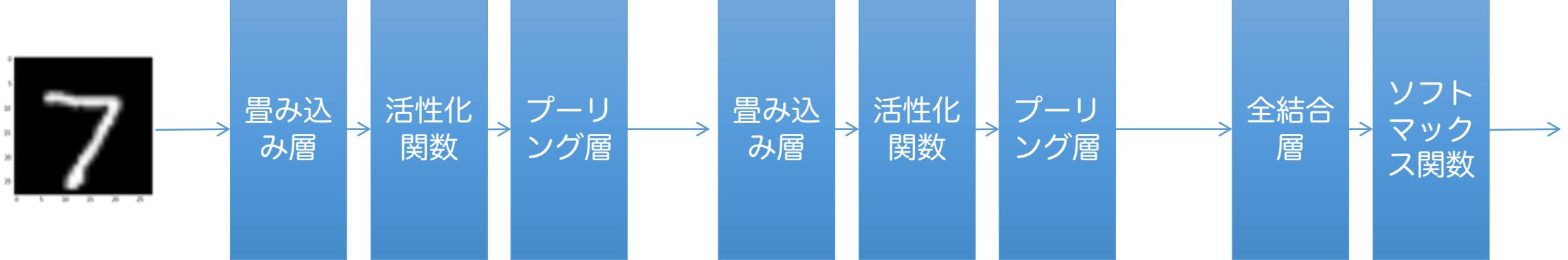
→ ニューラルネットワーク
(ディープラーニングなど) → 答え

全結合型NNとCNNの違い

全結合型NNの例



CNNの例



Any Questions?

畳み込み層

畳み込み演算

- ・ 畳み込み演算は、画像処理におけるフィルター演算に相当する。
- ・ フィルターという言葉は、文献によってはカーネルと表現されることもある。
- ・ フィルターの値は、NNの重みとして扱うことができ、学習によって最適な値が求められる。

× : 畳み込み演算

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

×

2	0	1
0	1	2
1	0	2

→

15	16
6	15

×

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

フィルター

2	0	1
0	1	2
1	0	2

→

15	

×

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

×

2	0	1
0	1	2
1	0	2

→

15	16

×

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

×

2	0	1
0	1	2
1	0	2

→

15	16
6	

×

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

×

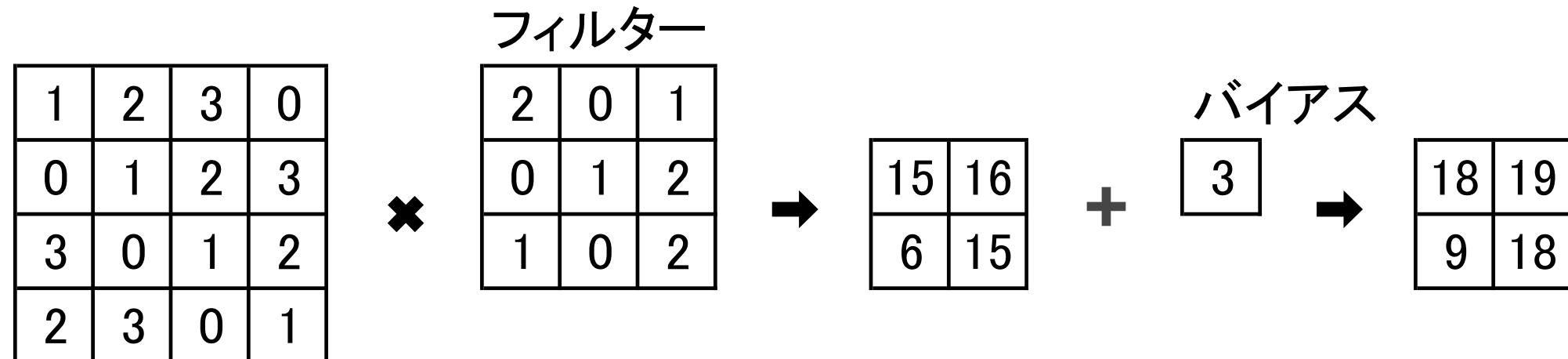
2	0	1
0	1	2
1	0	2

→

15	16
6	15

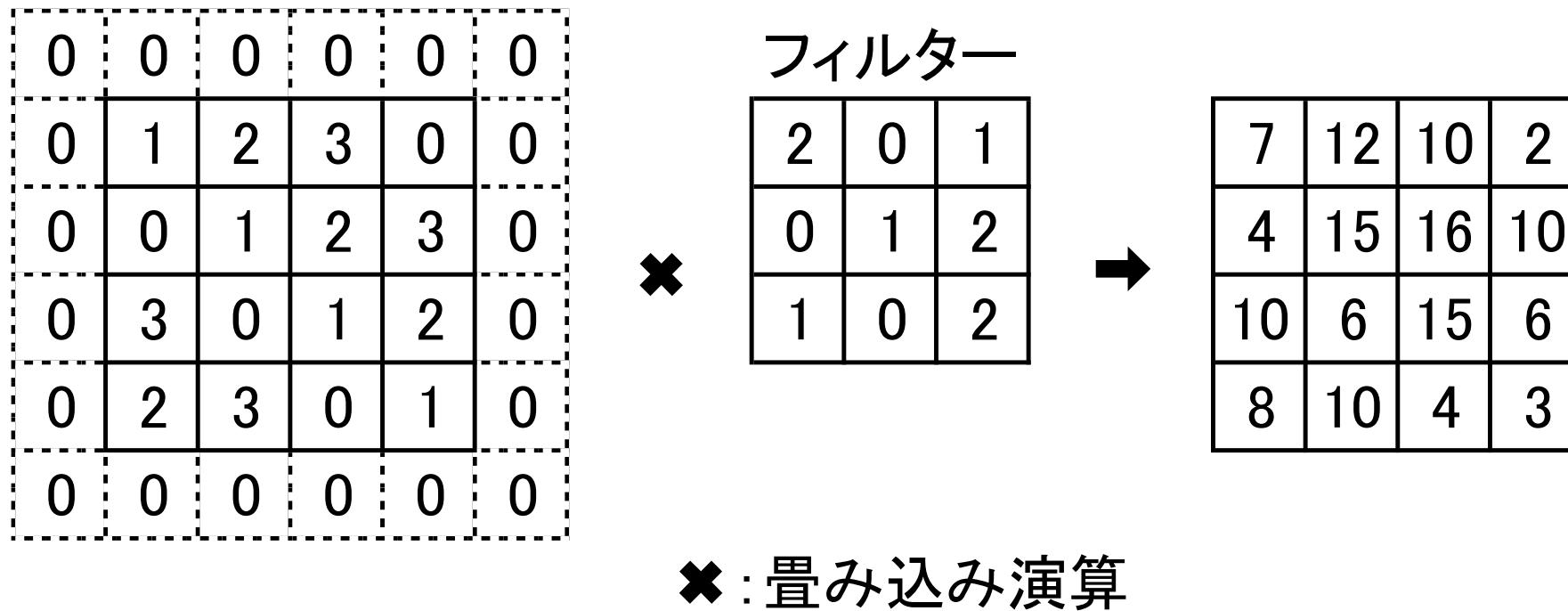
畳み込み演算のバイアス

- 畳み込み演算時のバイアスは、フィルター内で共通の1つのパラメータとして扱われることが多い。



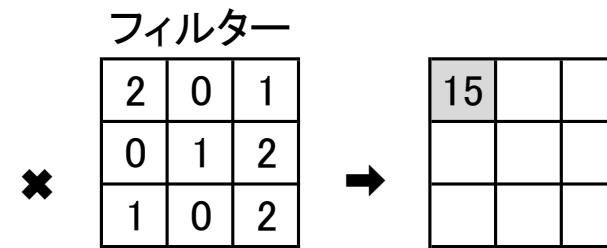
畳み込み演算のパディング処理

- ・ パディング処理とは、畳み込み演算を行う前に、画像の周辺に要素を追加すること。この要素は0で埋められることが多い(ゼロパディング)。
- ・ パディング処理を行うことによって、畳み込み演算後のサイズを元のサイズと同じにすることが可能。これにより、画像の周辺部の特徴を捉えやすくなる。



置み込み演算のストライド

1	2	3	0	1	2	3
0	1	2	3	0	1	2
3	0	1	2	3	0	1
2	3	0	1	2	3	0
1	2	3	0	1	2	3
0	1	2	3	0	1	2
3	0	1	2	3	0	1

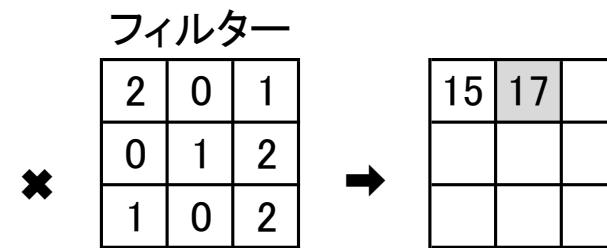


× : 置み込み演算

ストライド=2



1	2	3	0	1	2	3
0	1	2	3	0	1	2
3	0	1	2	3	0	1
2	3	0	1	2	3	0
1	2	3	0	1	2	3
0	1	2	3	0	1	2
3	0	1	2	3	0	1



× : 置み込み演算

[演習] 置み込み演算における出力サイズの計算

- ・ 置み込み演算における出力サイズは以下の式によって計算できます。
- ・ 手計算にて、前頁の出力サイズを計算してみましょう。

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

OH : 出力の高さ

OW : 出力の幅

H : 入力の高さ

W : 入力の幅

P : パディングサイズ

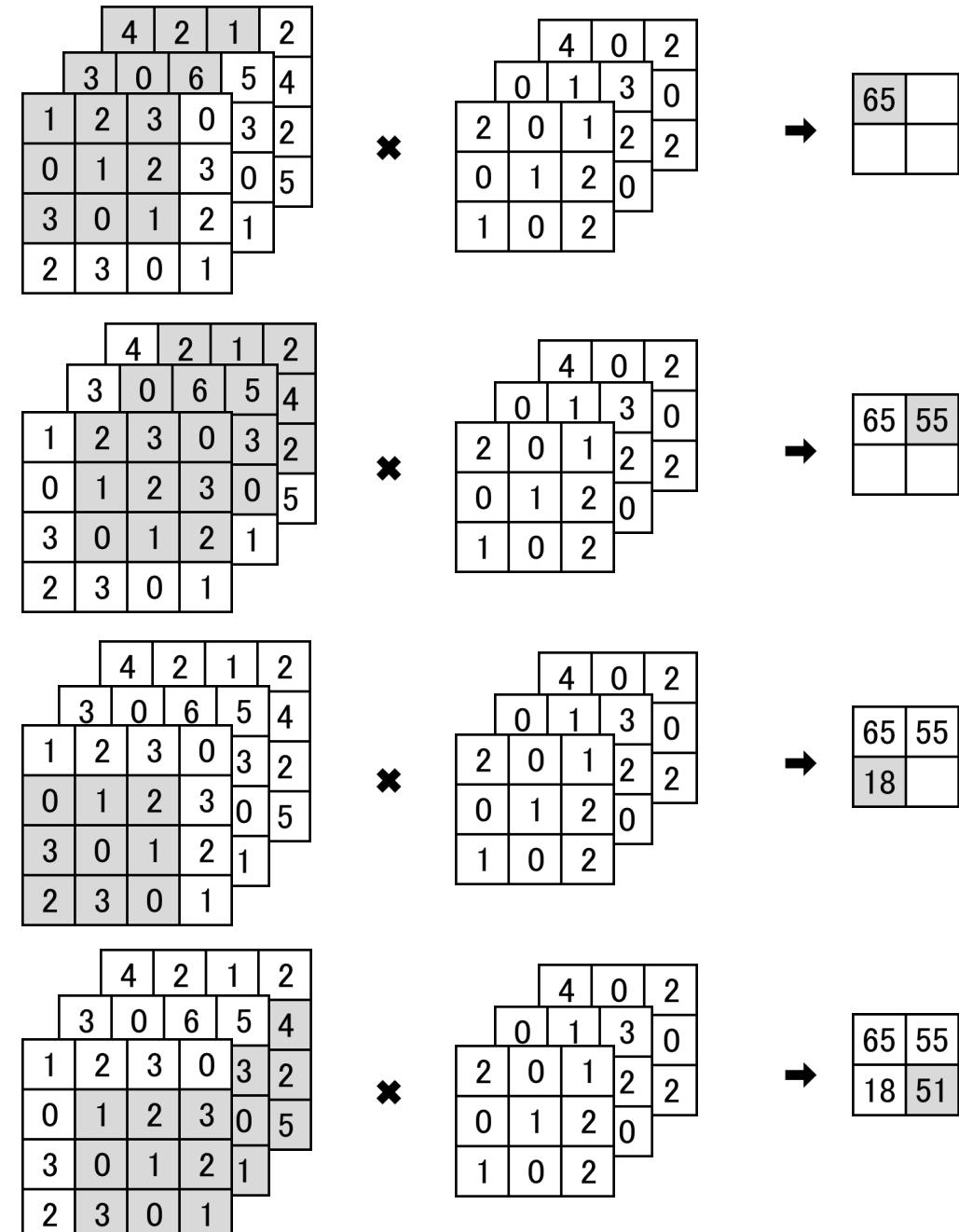
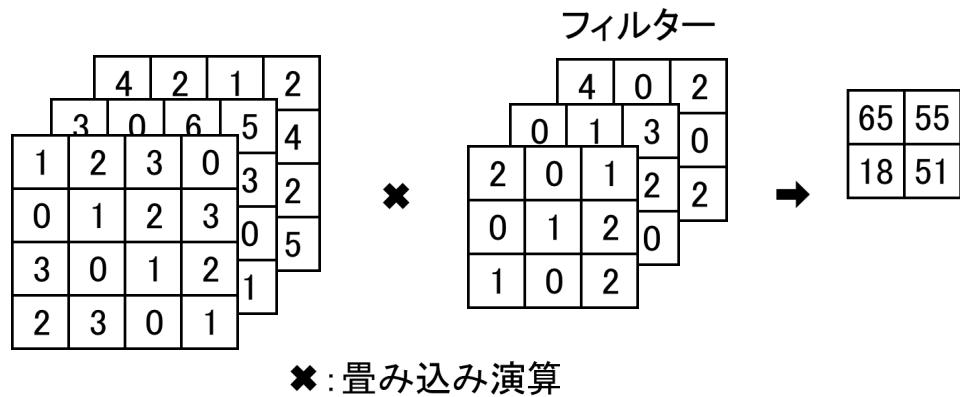
FH : フィルターの高さ

FW : フィルターの幅

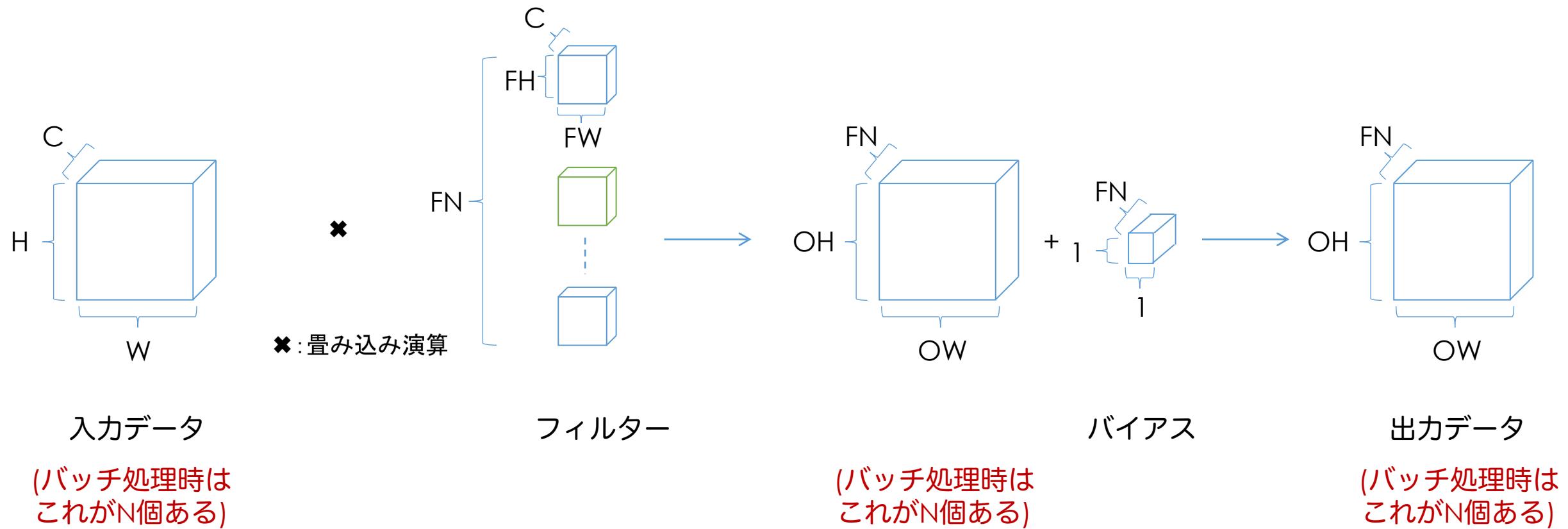
S : ストライドサイズ

3次元データ(3チャンネル)の畳み込み

- 複数のチャンネルがある場合、チャンネル毎に入力データとフィルターの畳み込みを行い、それらの結果を加算して1つの出力を得る。
- もしRGB毎の出力が有効である場合、フィルターの重みがそのように学習される。



置み込み演算をブロックで考える



[演習] 置み込み演算の仕組みを確認

- 5_1_convolution.ipynb
 - 置み込み演算の仕組みを確認しましょう。

Any Questions?

プリング層

プーリング演算

- ・ プーリング演算とは、要素を集約し、縦および横方向のサイズを小さくする演算である。
- ・ マックスプーリングや平均プーリングなどがある。
- ・ 畳み込み演算と異なり、学習する重みはない。

マックスプーリングの例

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



2	

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



2	3

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



2	3
4	

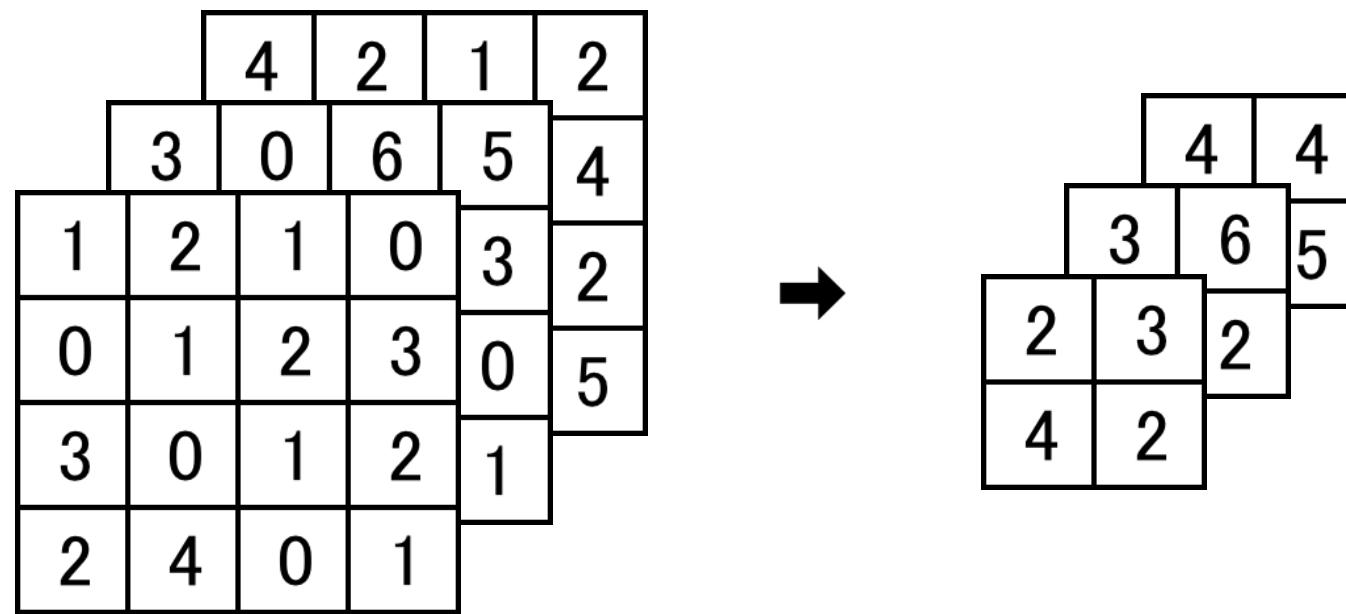
1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



2	3
4	2

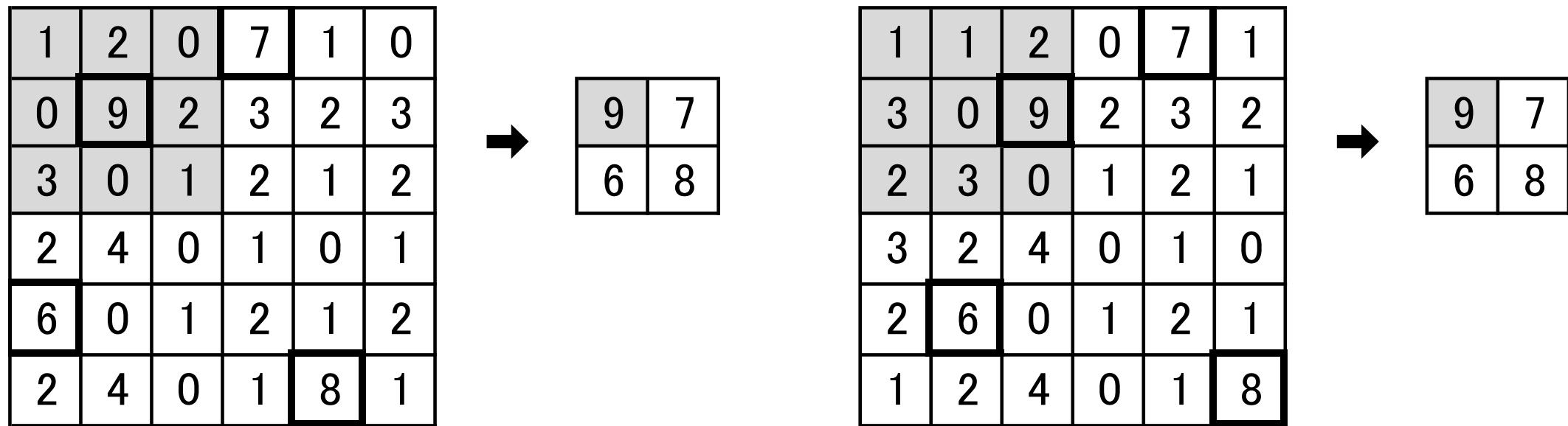
複数チャンネルデータのプーリング演算

- 畳み込み演算と異なり、プーリング演算を行なってもチャンネル数は変わらない。



プーリング演算の意義

- ・ プーリング演算によって画像のズレを吸収できる。
- ・ 例えば、以下のようなマックスプーリングでは、画像がずれてもプーリング後の出力が同じになる。



[演習] マックスプーリング演算の仕組みを確認

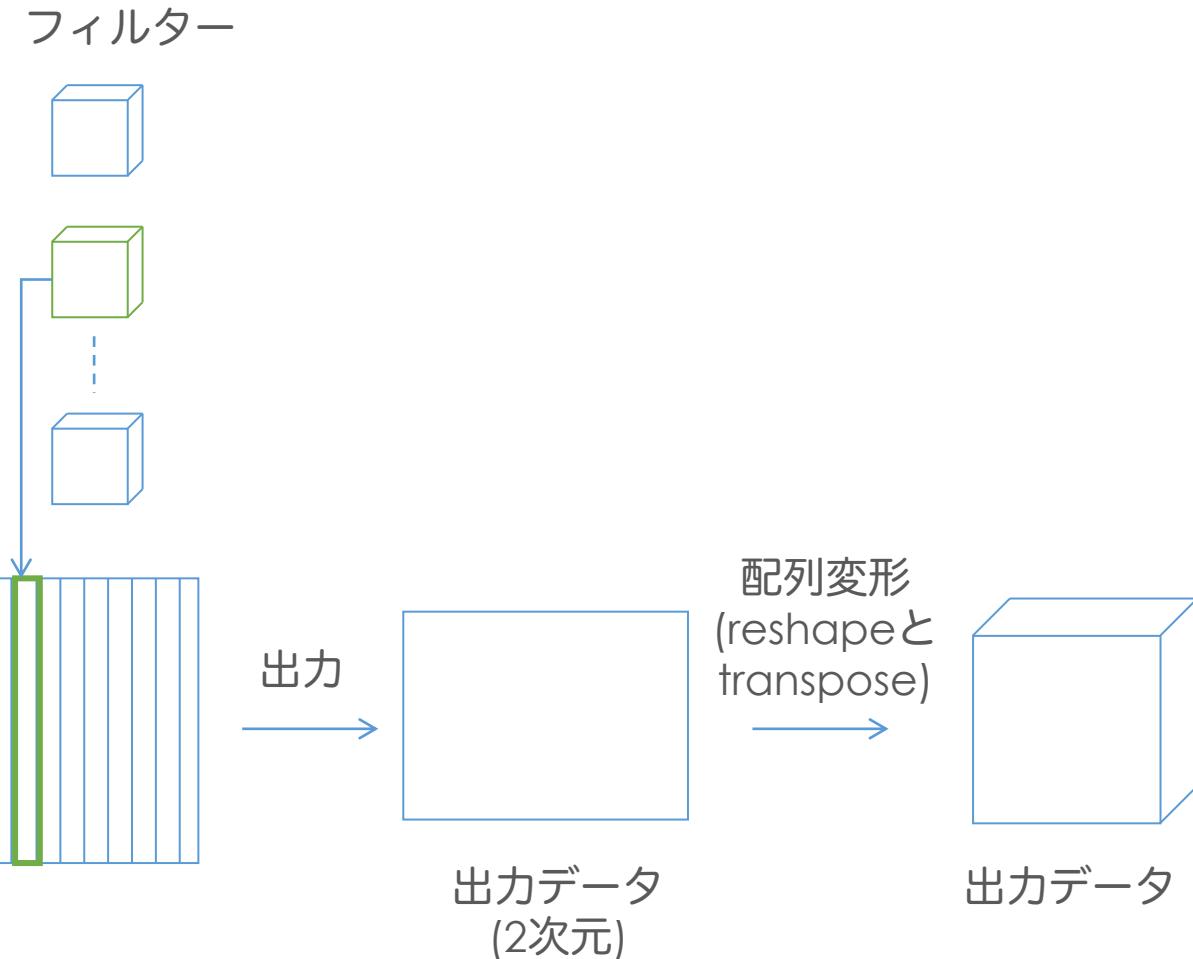
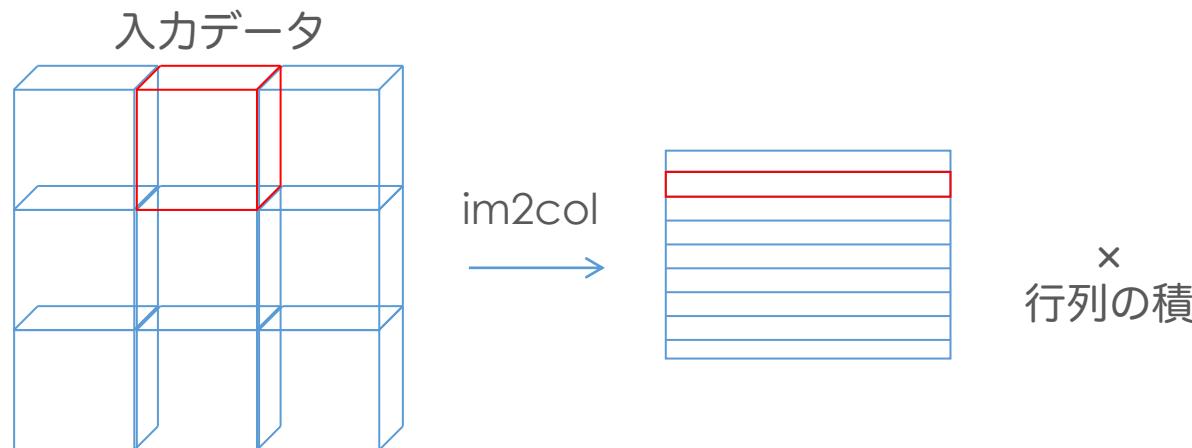
- 5_2_maxpooling.ipynb
 - マックスプーリング演算の仕組みを確認しましょう。

Any Questions?

im2col

im2col

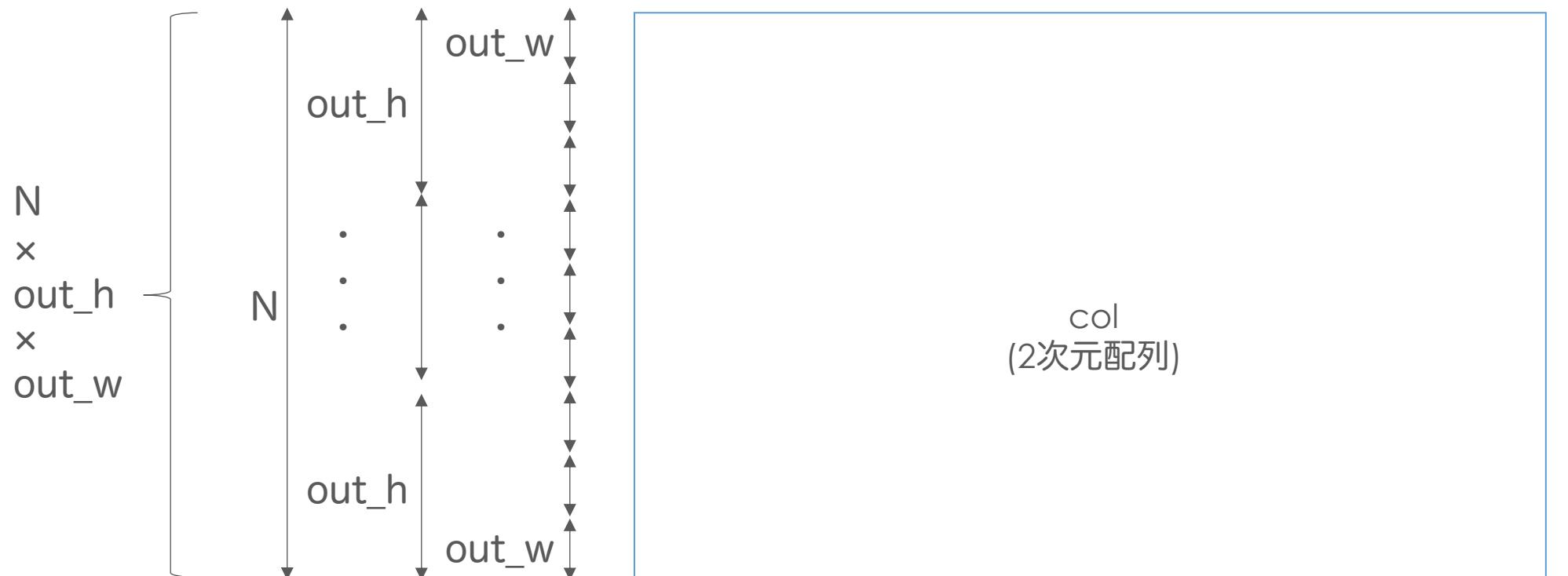
- im2col(image to column)は、畳み込み演算を効率的に行うためのアルゴリズム。
- 畳み込み演算を普通に計算しようとするとfor文が何重にもなる。im2colを用いると、for文が行列計算に置き換わり、演算が効率的になる。



im2col関数が返す配列

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0, constant_values=0):
    中略
    return col
```

- ミニバッチ学習時におけるim2col関数が返す配列colの構成を下図に示す。



プーリング演算時のim2col

- im2colは、プーリング演算にも用いることができる。

	4	2	1	2
3	0	6	5	4
1	2	3	0	3
0	1	2	4	0
1	0	4	2	1
3	2	0	1	

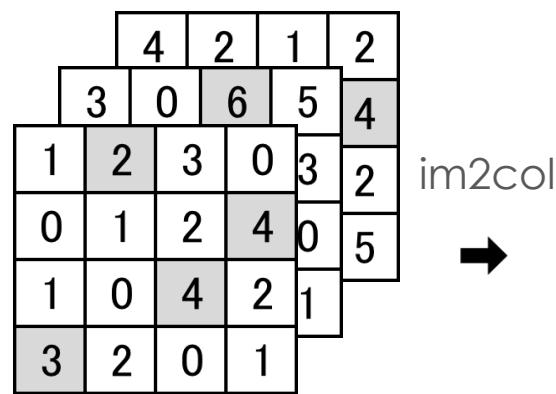
im2col
→

チャンネル軸を
行方向にもってくる
→

1	2	0	1
3	0	2	4
1	0	3	2
4	2	0	1
3	0	4	2
6	5	4	3
3	0	2	3
1	0	3	1
4	2	0	1
1	2	0	4
3	0	4	2
6	2	4	5

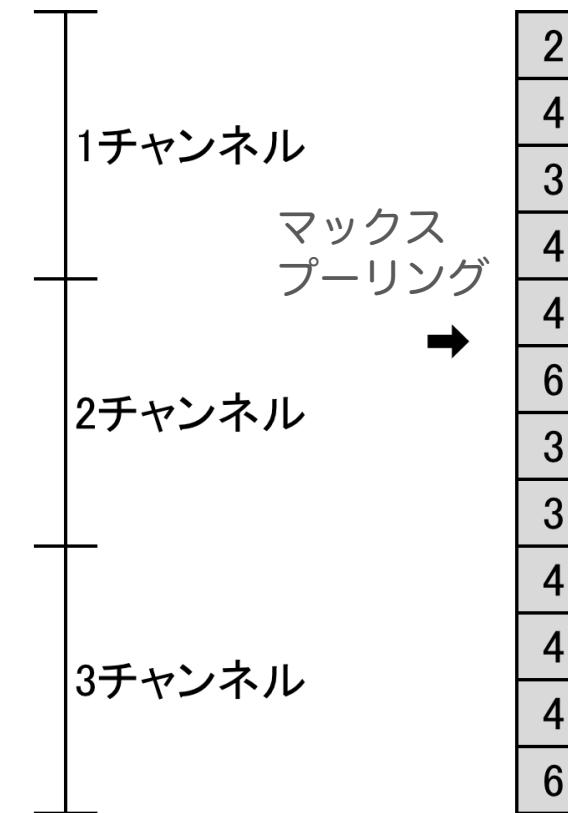


プーリング演算時のim2col

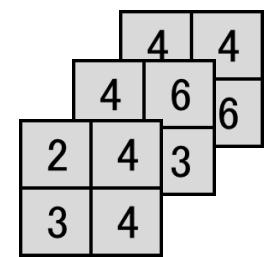


im2col
→
チャンネル軸
を行方向に
もってくる
→

1	2	0	1
3	0	2	4
1	0	3	2
4	2	0	1
3	0	4	2
6	5	4	3
3	0	2	3
1	0	3	1
4	2	0	1
1	2	0	4
3	0	4	2
6	2	4	5



配列変形
(reshapeと
transpose)



[演習] im2colとcol2imの実装

- 5_3_padding_reshape_tanspose.ipynb
 - im2colの実装に必要になる関数の挙動を確認しましょう。
- 5_4_im2col_col2im_trainee.ipynb
 - im2colとcol2imを実装しましょう。

[演習] 置み込み層クラスの実装

- 5_5_convolution_layer_trainee.ipynb
 - 置み込み層のクラスを実装しましょう。

[演習] マックスプーリング層クラスの実装

- 5_6_maxpooling_layer_trainee.ipynb
 - マックスプーリング層のクラスを実装しましょう。

[演習] シンプルなCNNの実装

- 5_7_simpleconvnet_trainee.ipynb
 - シンプルなCNNを実装し、MNIST問題を解いてみましょう。

Any Questions?

データ拡張

データ拡張

- 学習用データを加工複製し、それを学習用データに加えて学習させることをデータ拡張(augmentation)と呼ぶ。
- 特に、CNNの学習でよく利用される。
- 拡張方法の例
 - 回転
 - 反転
 - 伸縮
 - ノイズ
 - 部分的に隠す
- 参考ライブラリ
 - <https://github.com/mdbloice/Augmentor>

[演習] データ拡張の試行

- 5_8_augmentation.ipynb
 - データ拡張を試行してみましょう。

[グループワーク] カタカナ15文字のデータ拡張

- ・ カタカナ15文字の学習用データを用いて様々なデータ拡張を行ってみましょう。
- ・ カタカナ15文字を分類するモデルを学習させる際は、どのようなデータ拡張が向いているでしょうか？グループで話し合ってみましょう。
- ・ 2~3名のグループに分かれて、上記課題に取り組みましょう。(30分)
- ・ 最後に、グループごとに発表していただきます。 (15分)

Any Questions?

CNNの様々なモデル

目次

1. 著名なCNNモデル
2. 物体検出タスクとCNN
3. セマンティックセグメンテーションタスクとCNN

著名なCNNモデル

LeNet-5

- LeNet-5は、6層構造(CNN層が5つ)のCNN。
- 32×32ピクセルのパッチ画像を入力とする。
- 1998年ごろYann LeCunらによって提案された。
 - <http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf>
 - <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>
- 畳み込み層の考え方は、1989年ごろから提案されていた。
 - <http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf>

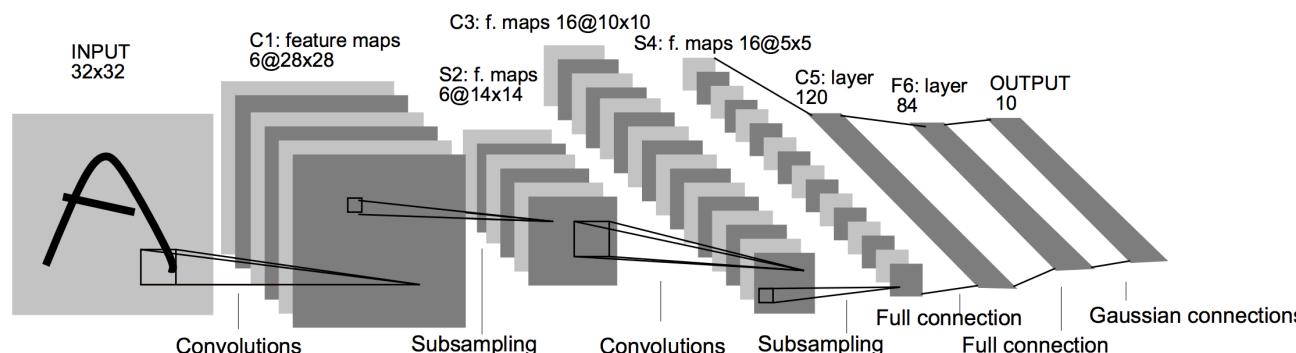
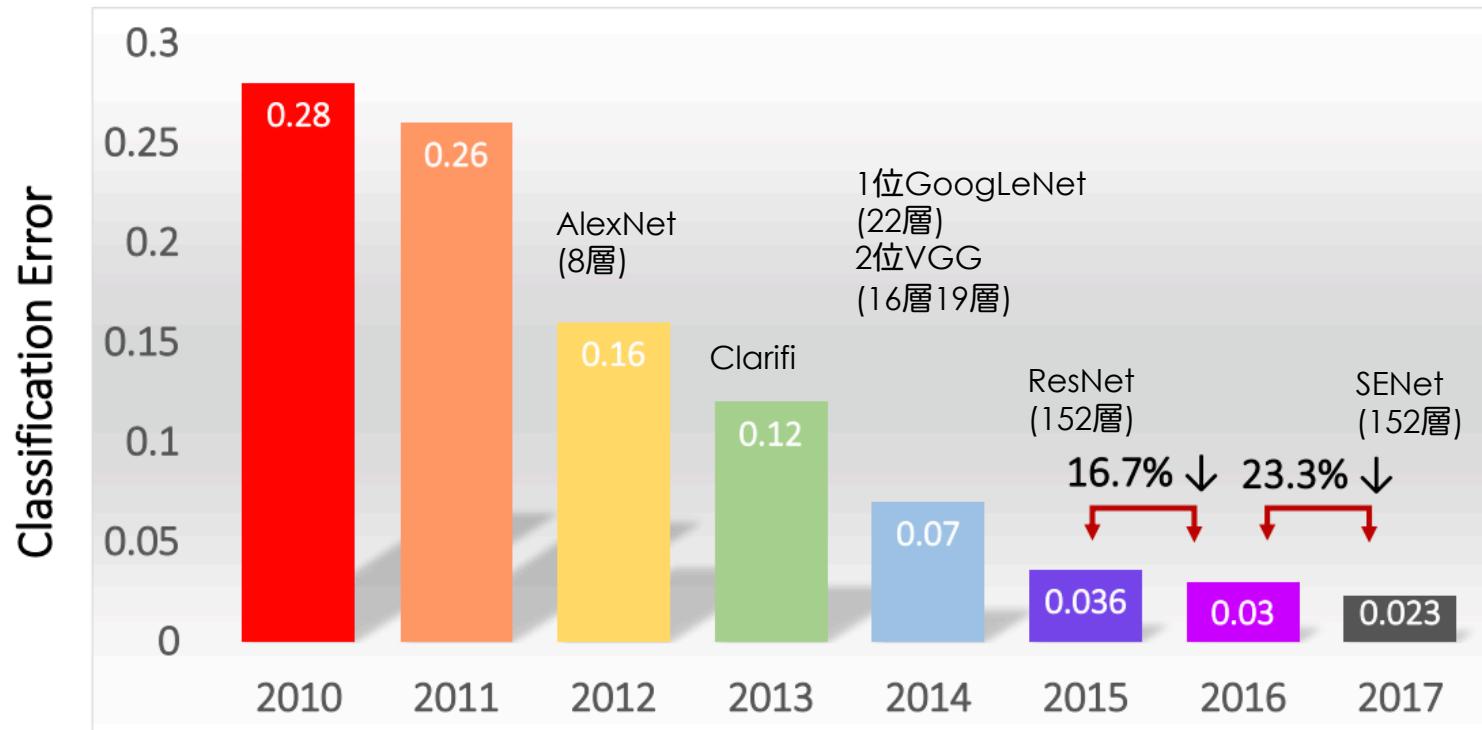


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNetのネットワーク

ILSVRC

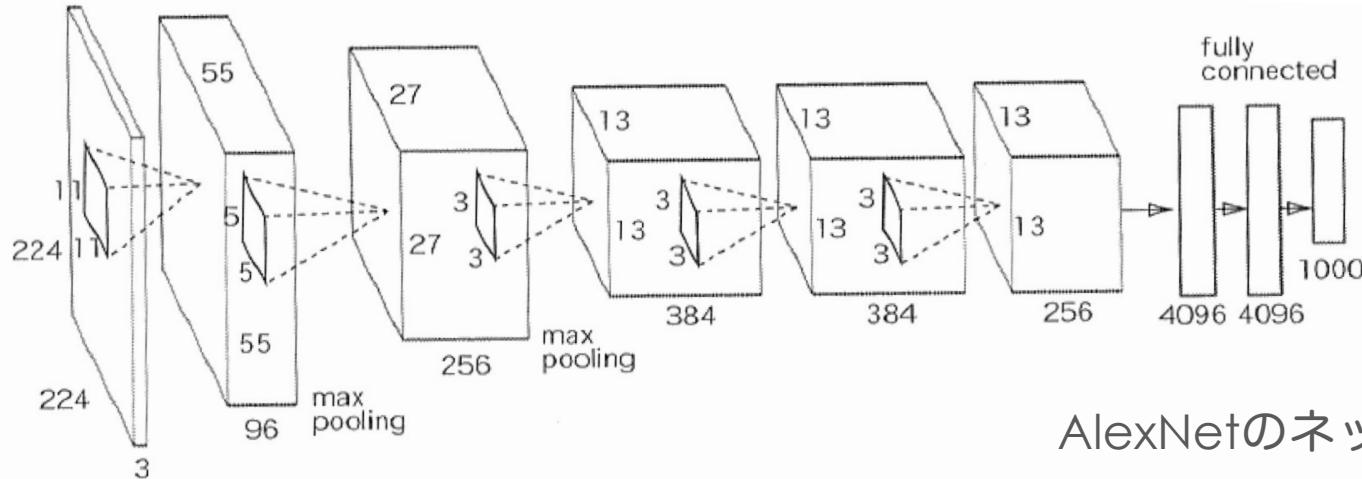
- ILSVRC (Image net Large Scale Visual Recognition Challenge)は、2010年から始まった大規模画像認識のコンテスト。



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

AlexNet

- 2012年のILSVRCで1位になったネットワーク。
- 5層の畳み込み層と3層の全結合層で構成される。
- パラメータ数は、約6000万個。
- <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

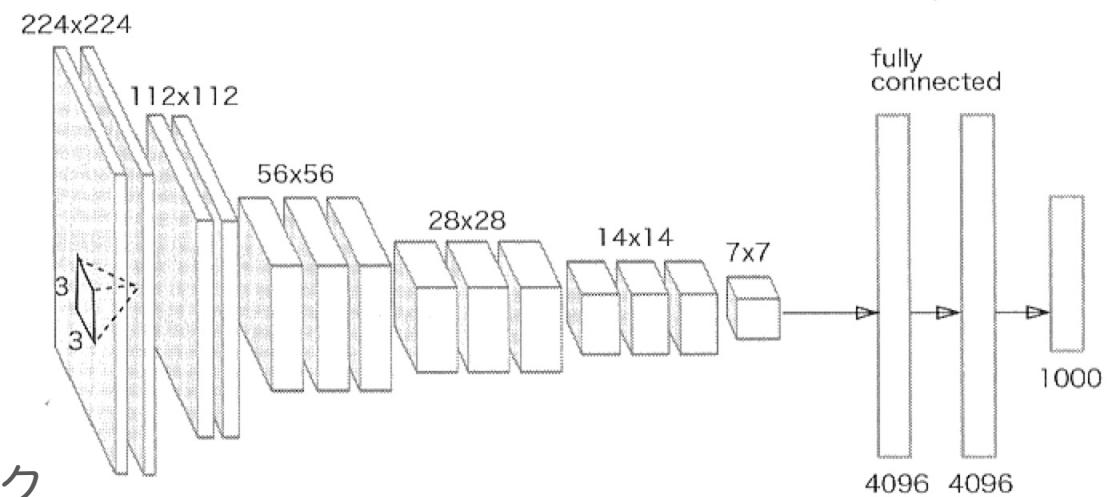


AlexNetのネットワーク

VGG

- 2014年のILSVRCで2位になったネットワーク。
- 19層のネットワークはVGG-19、16層のネットワークはVGG-16と呼ばれる。
- AlexNetよりも深いネットワーク構造。
- VGGとは、Visual Geometry Group(Oxford大学の研究室)の略。
- パラメータ数は、約1億4000万個。
- 構造がシンプルなため、実務でよく使われる。
- <https://arxiv.org/pdf/1409.1556.pdf>

VGG-16のネットワーク



GoogLeNet

- 2014年のILSVRCで1位になったネットワーク。
- <https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>

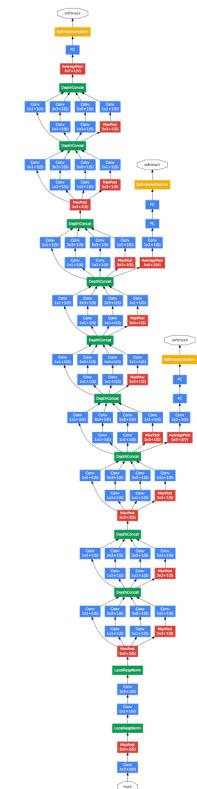
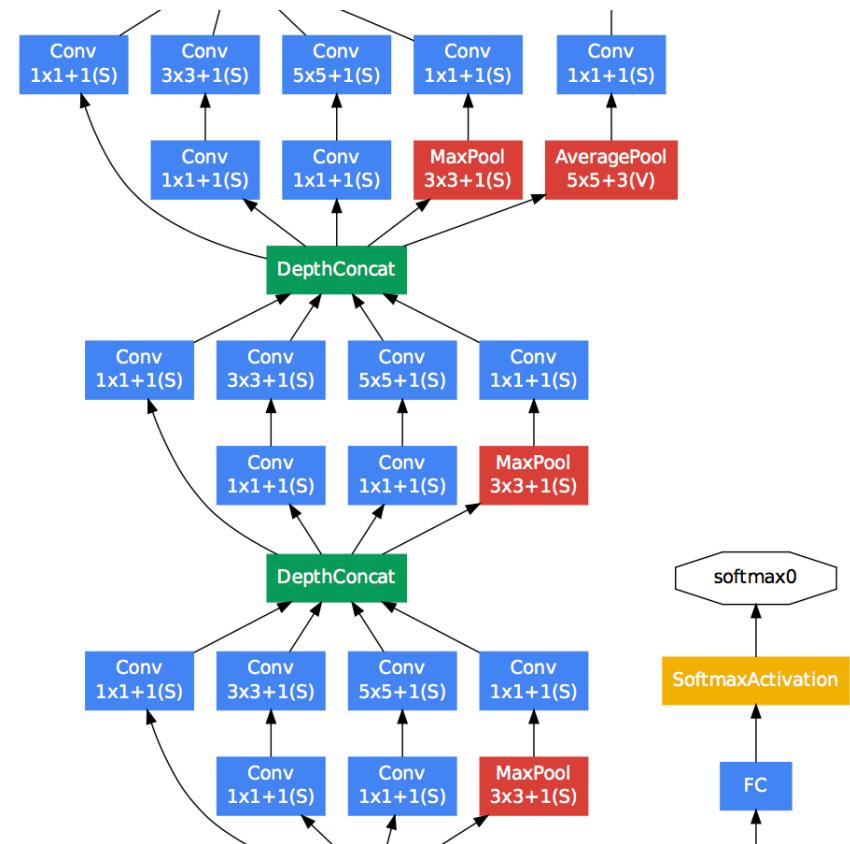
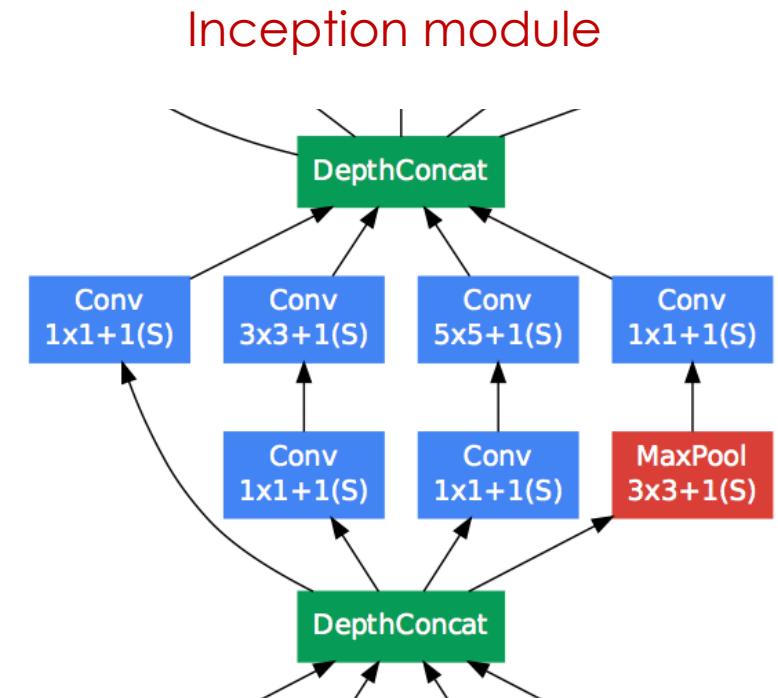


Figure 3: GoogLeNet network with all the bells and whistles



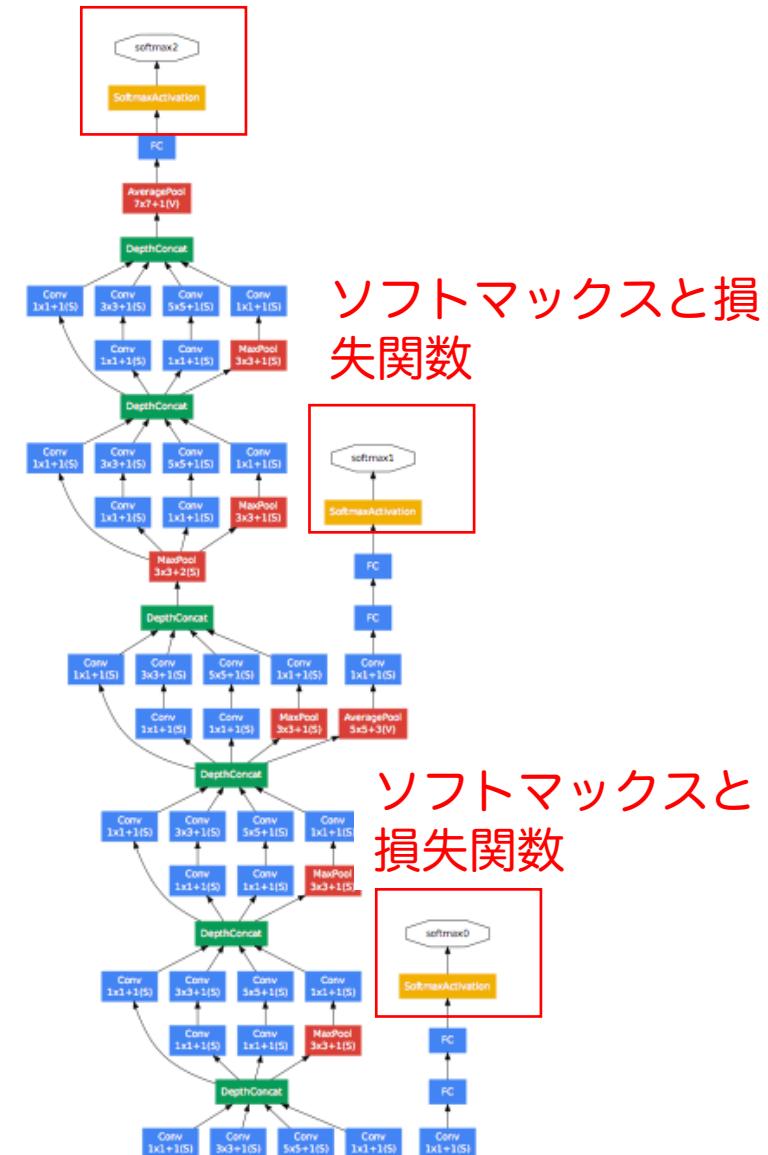
GoogLeNetの仕組み

- Inception moduleと呼ばれる複数のフィルタ群により構成されたブロックが特徴。
- Inception moduleは、小さなネットワークから構成される。GoogleNetは、このモジュールを積み重ねた構成になっている。
- Inception moduleは、**小さな畳み込みフィルタを並列に並べることで、より少ないパラメータで同等の表現力を実現することができる**。
- Inception moduleには 1×1 の畳み込みフィルタが使われているが、このフィルターは次元削減と等価な効果がある。



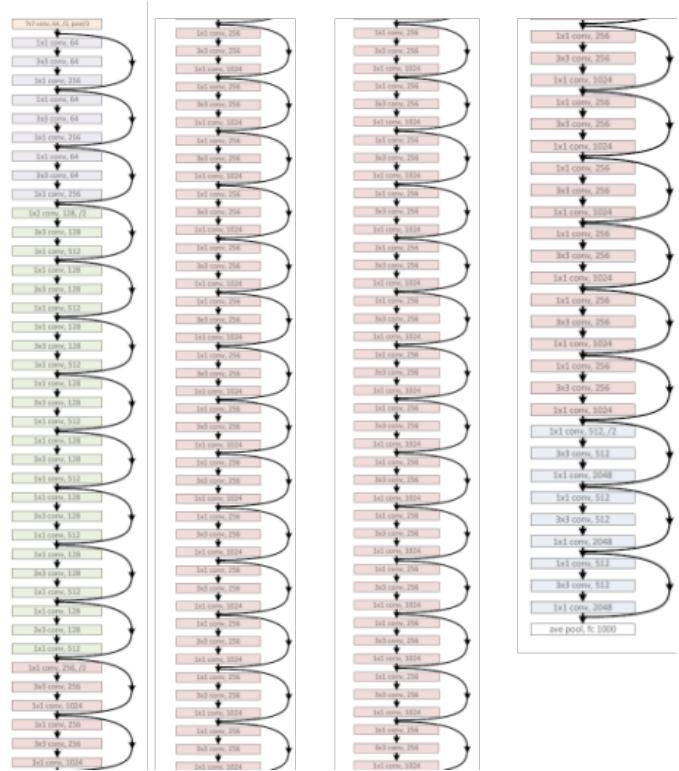
GoogLeNetの仕組み

- GoogLeNetのもう一つの特徴として、補助的損失(auxiliary loss)がある。
- GoogLeNetの学習ではネットワークの途中から分岐させたサブネットワークにおいてもクラス分類を行っている。
- ネットワークの中間層に直接誤差を伝播させることで、勾配消失を抑えることができる。
- 複数の損失関数があることにより、アンサンブル学習と同様の効果が得られるため、汎化性能の向上が期待できる。ただし、予測時は最終層の出力結果だけを用いる。

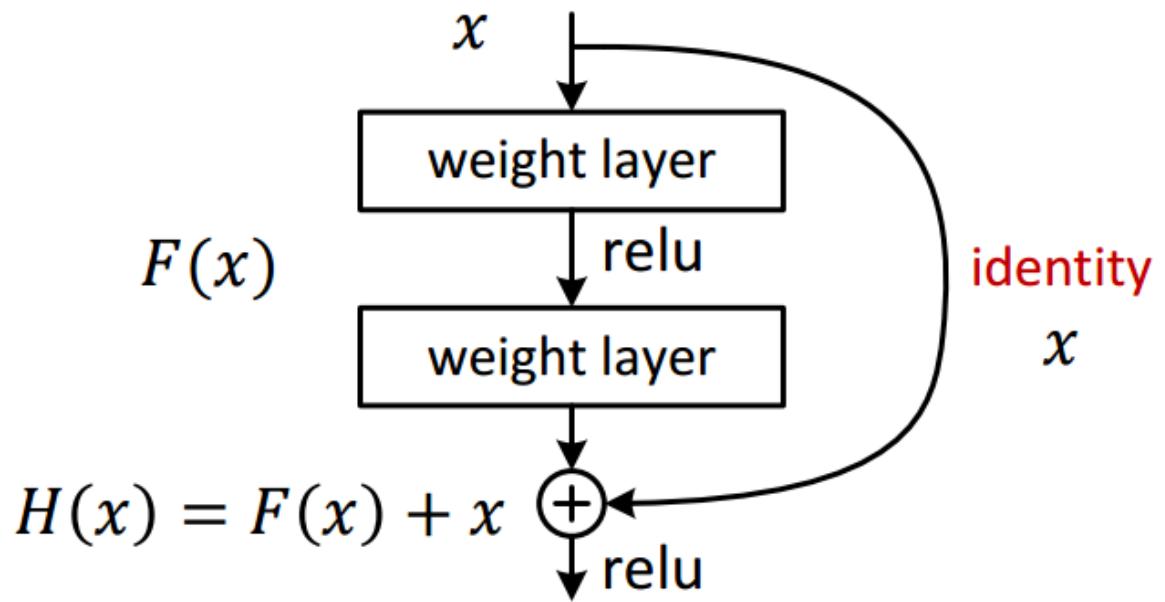


ResNet

- 2015年のILSVRCで1位になったネットワーク。
- Microsoftのチームのよって開発された。
- <https://arxiv.org/pdf/1512.03385.pdf>



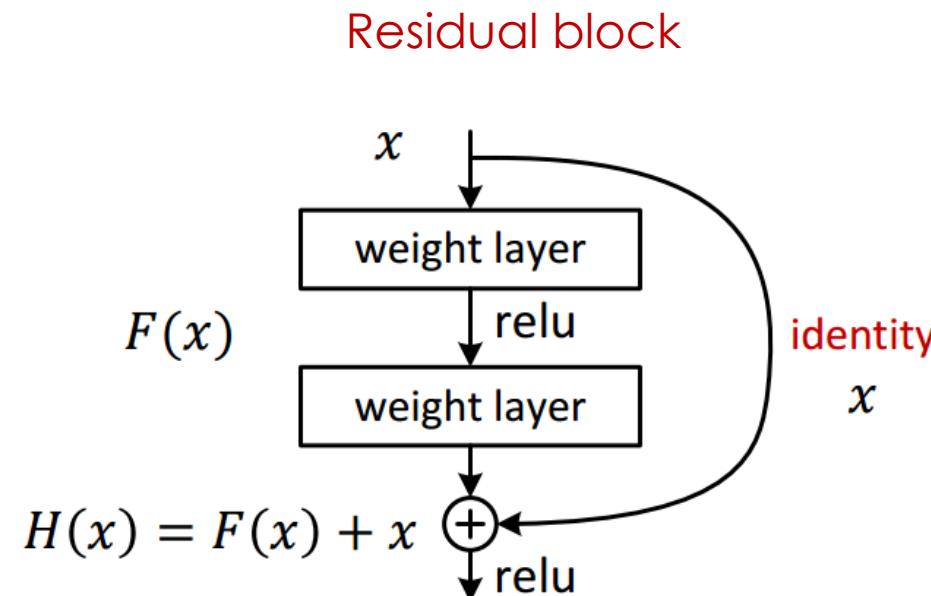
• Residual net



ResNetの仕組み

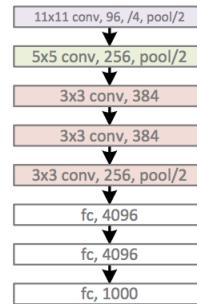
- ResNetは、Residual blockというユニットが連結された形になっている。
- Residual blockには、層をまたがる結合(Identity mapping)がある。このショートカット結合により、逆伝播計算時に勾配が減衰していくことを抑えることができる。
- この手法により、従来の「層を深くすると学習できない」という問題が解決された。

$F(x)$ では、入力 x と出力 $H(x)$ の差(残差)である $H(x) - x$ を学習しているため、Residual block(残差ブロック)と呼ばれる

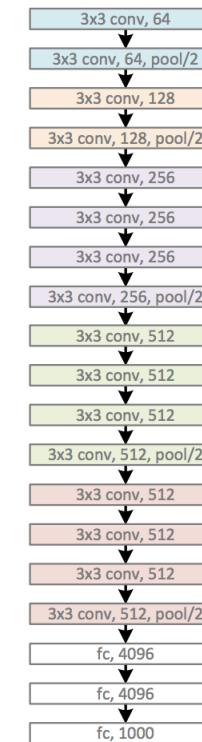


Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)



DenseNet

- ResNetに似たショートカット結合をもつネットワーク。
- 特徴量を再利用する結合になっている。
- DenseNetの特徴
 - ショートカット結合を用いているため、勾配消失が起きにくい。
 - DenseNet以前のネットワークよりも、パラメータ数を少なくすることができる。
- 原著論文
 - G.Huang, Z.Liu, L.van der Maaten, K.Q.Weinberger. Densely Connected Convolutional Networks. IEEE Conference on Pattern Recognition and Computer Vision (CVPR), 2016., <https://arxiv.org/pdf/1608.06993.pdf>

DenseNet

- DenseNetは、DenseBlockによって構成される。

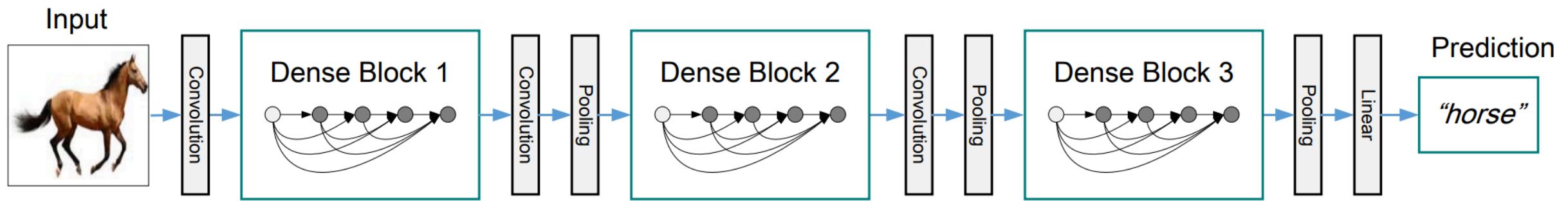
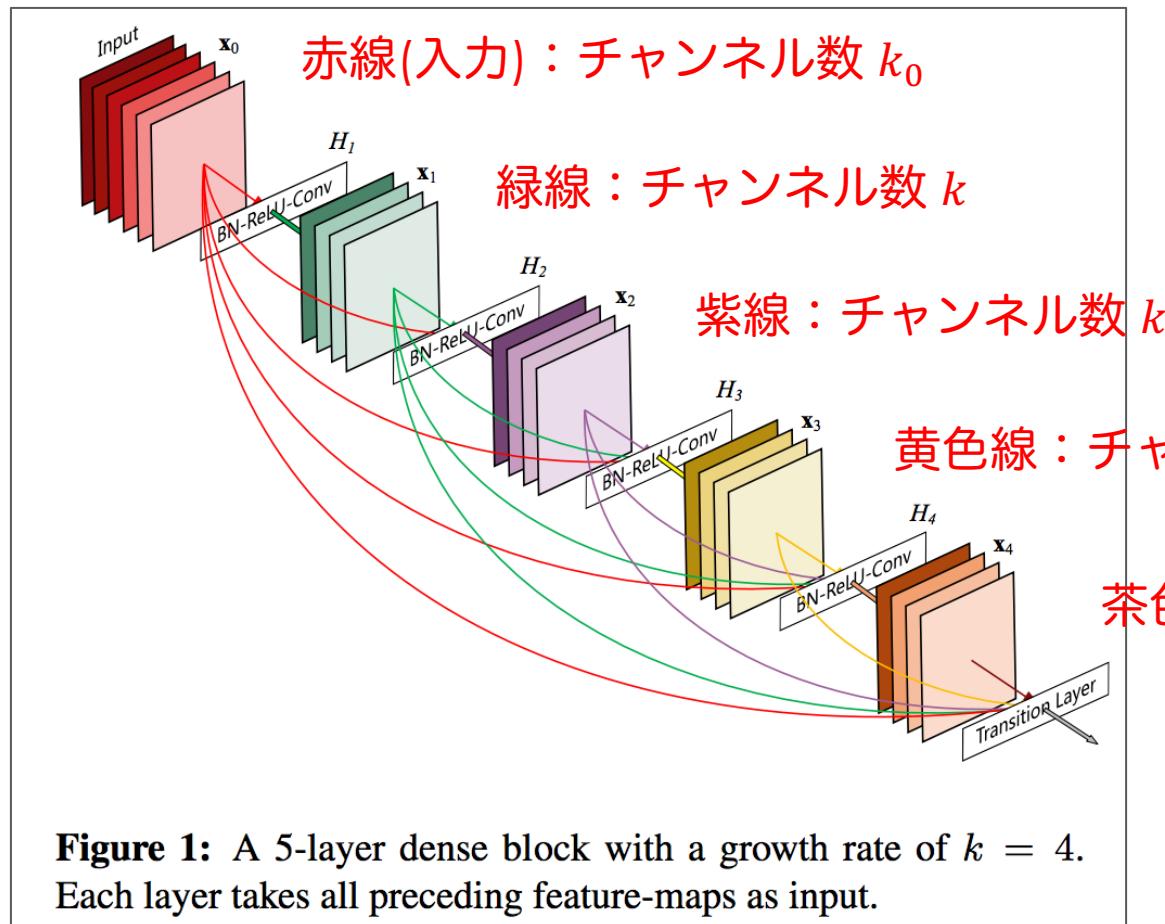


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

引用元: <https://arxiv.org/pdf/1608.06993.pdf>

DenseNet

- DenseBlockの結合方法を5層の場合で示す。



チャンネル数 k は成長率(Growth rate)と呼ばれ、ハイパーパラメータになる。

BN-ReLU-Convにおいて、異なる色の線は、チャンネル方向に結合される

茶色線 : チャンネル数 k

白線(出力) : チャンネル数 k_0+4k

※モデルを小さくするために、transition layerにて、出力チャンネル数を k_0+4k よりも小さくさせる場合がある

引用元: <https://arxiv.org/pdf/1608.06993.pdf>

Any Questions?

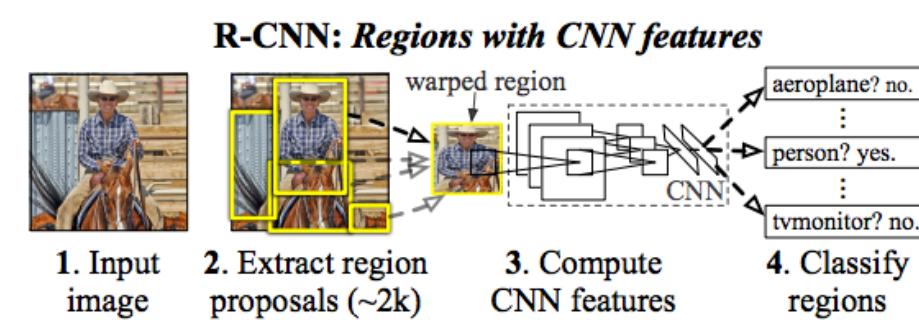
物体検出タスクとCNN

物体検出タスク

- ・ 物体検出タスクとは、ある画像中における物体の位置を検出し、それにラベルを付与するタスクのこと。
- ・ 物体検出で使われる主なアルゴリズムを以下に示す。
 - ・ R-CNN
 - ・ Fast R-CNN
 - ・ Faster R-CNN
 - ・ YOLO(you only look once)
 - ・ 2019年5月時点でv1~v3が発表されているが、ここではv1について紹介する。
 - ・ SSD(single shot multibox detector)
- ・ 物体検出タスクでは、物体を囲う矩形のことをバウンディングボックスと呼ぶ。

R-CNN

- R-CNN(regions with CNN features)は、物体検出方法の1つ。
- 計算手順
 1. 入力画像に対して、物体が写っている領域の候補を選択的検索法(selective search method)で約2,000個抽出する。
 2. CNN のインプットの大きさに合うようにそれぞれの領域中の画像をリサイズする。
 3. それぞれの物体領域候補に対して、CNN (原著論文では AlexNet) で特徴マップを計算する。
 4. それぞれの領域に何が写っているか SVM で分類する。
 5. 物体の詳細位置を決めるために、特徴マップとバウンディングボックス座標を回帰させる問題を解く。
- 約2000画像の特徴を抽出するため、計算に時間がかかる。



引用元：Rich feature hierarchies for accurate object detection and semantic segmentation, Ross Girshick. et al.

<https://arxiv.org/pdf/1311.2524.pdf>

選択的検索法

- 選択的検索法(selective search method)とは、ピクセルレベルで類似する領域をグルーピングしていくことで候補領域を選出するアルゴリズム。
- 似たような特徴を持つ領域を結合していく、1つのオブジェクトとして抽出する。
- 計算手順
 - 画像を小さい領域群に分割する。
 - 隣り合う領域どうしの類似度を計算する。
 - 最も類似度の高い領域を統合する。
 - 領域が1つになるまで、2と3を繰り返す。



Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

引用元： Selective Search for Object Recognition, J.R.R. Uijlings et al
<https://koen.me/research/pub/uijlings-ijcv2013-draft.pdf>

Fast R-CNN

- R-CNNは、提案された物体領域候補ごとに順伝播計算を行う必要があり、検出速度の遅いことが課題であった。
- 画像全体を入力して計算した特徴マップをすべての物体領域候補で再利用することで、R-CNNを高速化させたものがFast R-CNN。
- 計算手順
 1. 選択的検索法を利用して物体領域候補を算出する。
 2. 画像全体をCNNに入力して特徴マップを計算する。
 3. 物体領域候補ごとに以下の計算を行う。
 1. 物体領域候補に対応する特徴マップを抽出する。
 2. 特徴マップをRoI(Region of Interest)プーリング層に渡し、次元数の異なる特徴マップを一定の大きさのベクトルに変換する。
 3. 物体の詳細位置を決めるために、特徴マップとバウンディングボックス座標を回帰させる問題を解く。

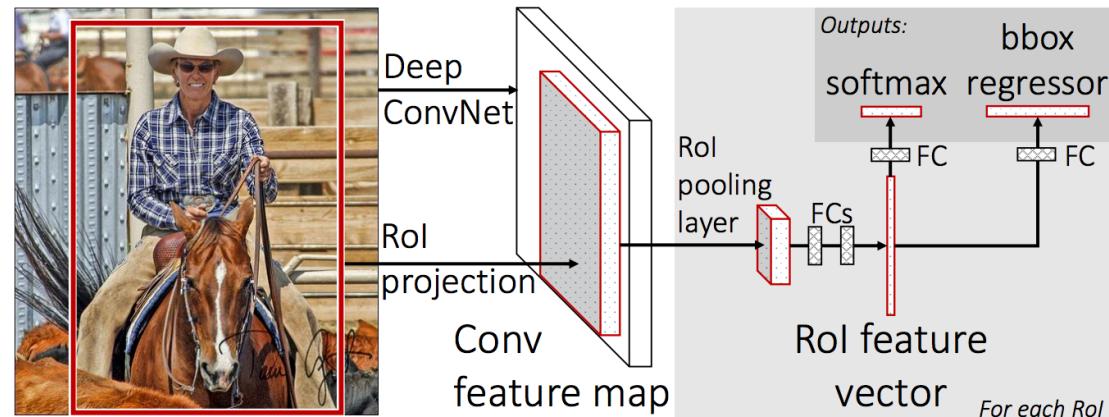


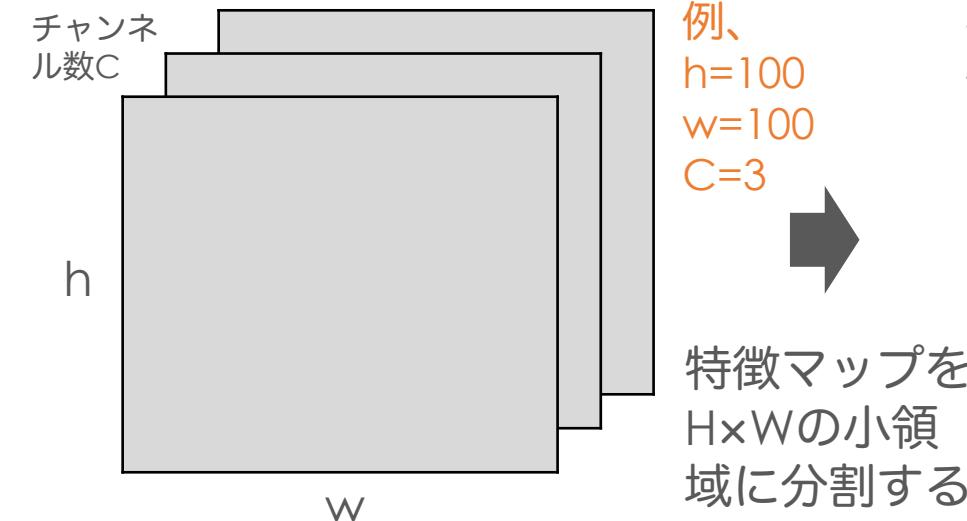
Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

引用元：Fast R-CNN, Ross Girshick,
<https://arxiv.org/pdf/1504.08083.pdf>

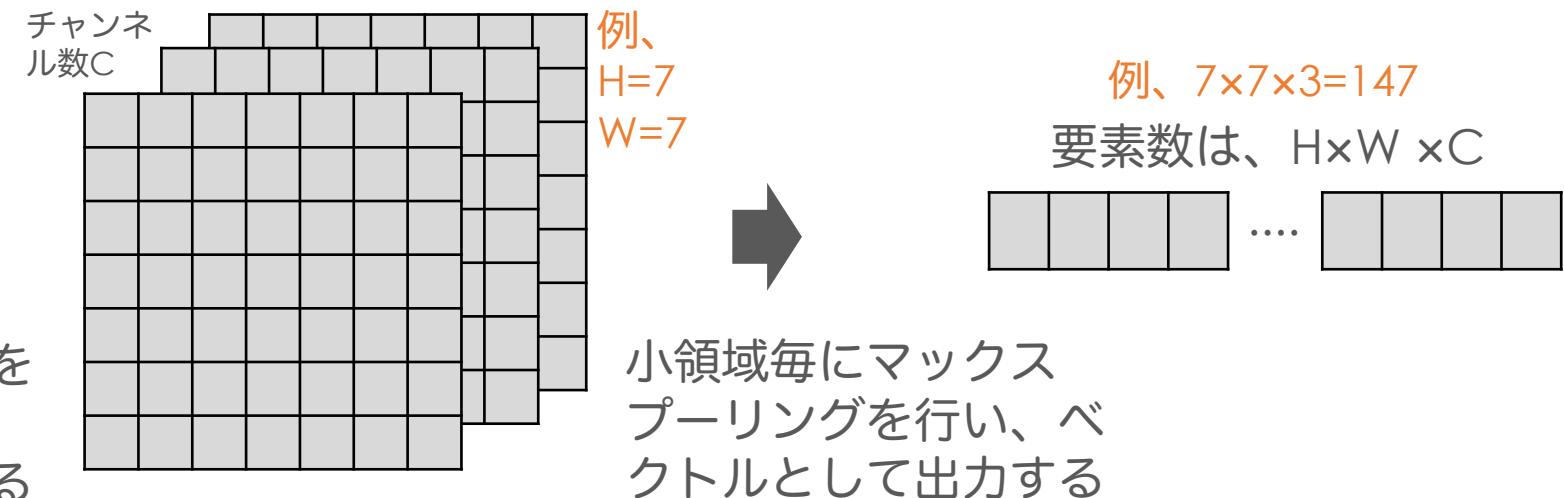
RoIプーリング層

- RoI(Region of Interest)プーリング層では、物体領域候補の特徴マップが入力され、マックスプーリング演算を行い、要素数が $H \times W \times C$ のベクトルが出力される。HとWはあらかじめ決めておく分割数(例、 $H=7$ $W=7$)であり、Cは特徴マップのチャンネル数である。
- $H \times W$ のそれぞれの領域について、 1×1 の空間ピラミッドプーリングを行なっていることに相当する。

特徴マップ($h \times w \times C$)



RoIプーリング層の出力



空間ピラミッドプーリング

- 空間ピラミッドプーリング(SPP, spatial pyramid pooling)とは、Microsoft Research Asiaが2014年に開発した新しいプーリング層。
- 入力画像のサイズが異なっている場合でも、出力のサイズがいつも同じになるのが特徴。
- 仕組みは単純で、画像を格子状に、 1×1 、 2×2 、 4×4 、 16×16 、...と分割し、その中で、マックスプーリング(ほかのプーリングでもいい)を行う。その後、 1×1 の出力+ 2×2 の出力+ 4×4 の出力+ 16×16 の出力...と、つなげたベクトルをSPPの出力とする。
- 出力のサイズはピラミッドのサイズとチャンネル数(Fig3では256)にのみによって決まる。
- 参考：http://image-net.org/challenges/LSVRC/2014/slides/sppnet_ilsvrc2014.pdf

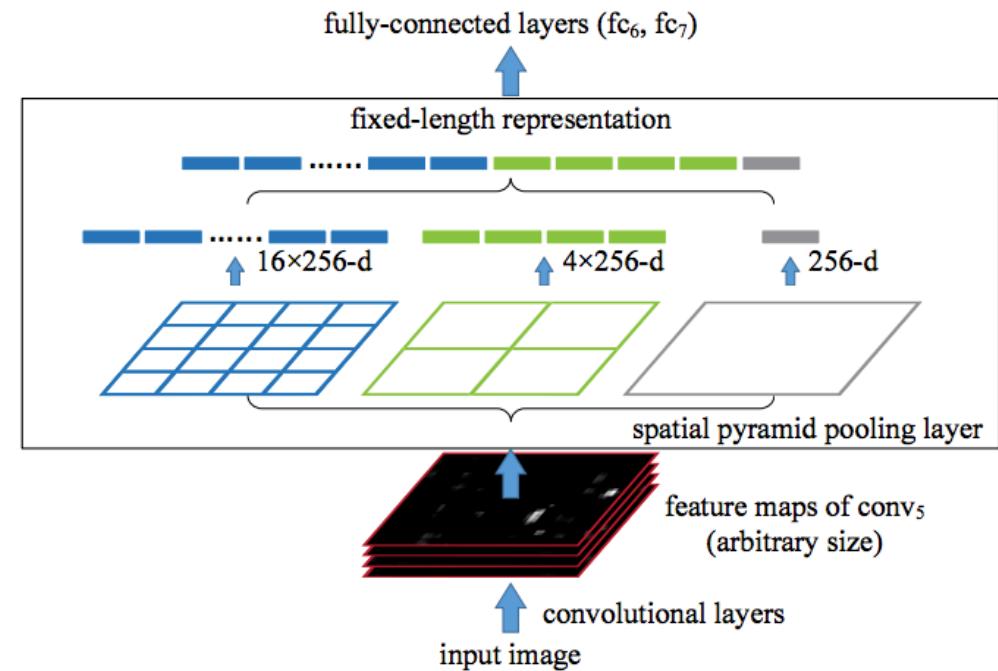


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the conv₅ layer, and conv₅ is the last convolutional layer.

引用元：Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,,Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun
<https://arxiv.org/pdf/1406.4729v4.pdf>

Faster R-CNN

- Fast R-CNNでは、ニューラルネットワークとは別のモジュールで物体領域候補を計算する必要があった。
- Faster R-CNNは、特徴マップから物体領域候補を推定する領域提案ネットワーク(RPN, region proposal network)とFast R-CNNの2つのネットワークで構成される。
- 2つのネットワークで特徴マップを共有しているため、計算が効率的。
- Fast R-CNN側のネットワークで、特徴マップとバウンディングボックス座標を回帰させる問題を解く。

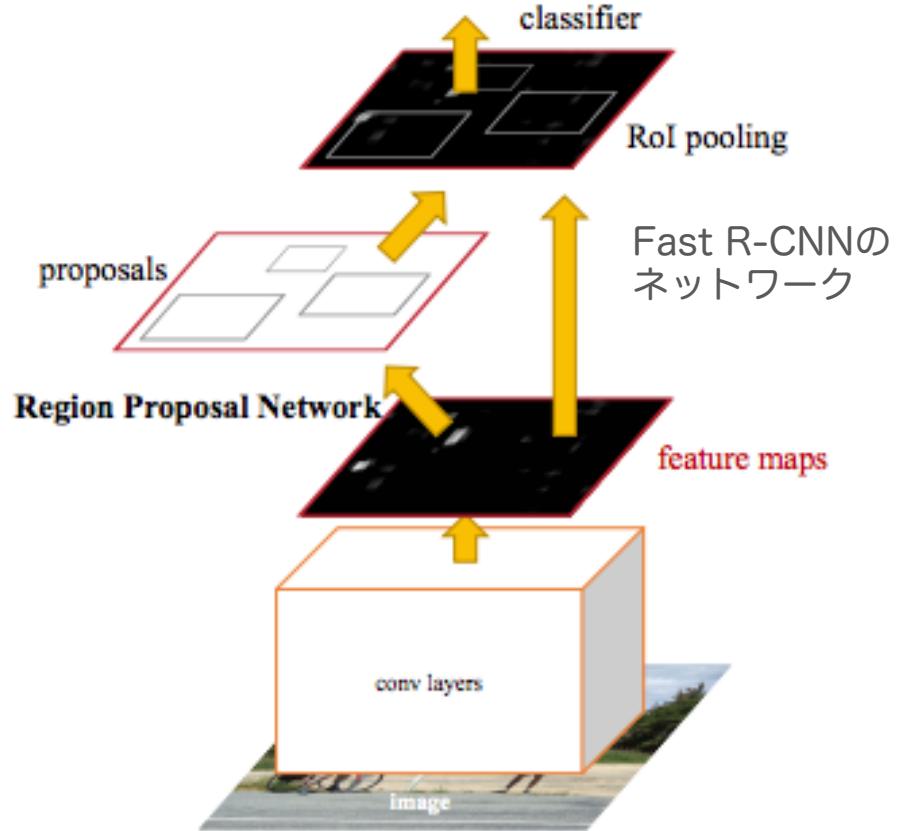


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

引用元 : Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ,
Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun ,
<https://arxiv.org/pdf/1506.01497.pdf>

YOLO

- YOLO (you only look once)は、画像全体をグリッドに分割(分割された領域をセルと呼ぶ)し、セル毎にクラスとバウンディングボックスを求める方法。
- 1つのニューラルネットワークで、クラス分類問題とバウンディングボックス回帰問題の両方を解く。
- Faster R-CNN に比べ、CNNのアーキテクチャがシンプルになった分、検出速度は高速になるが、精度は少し劣る。

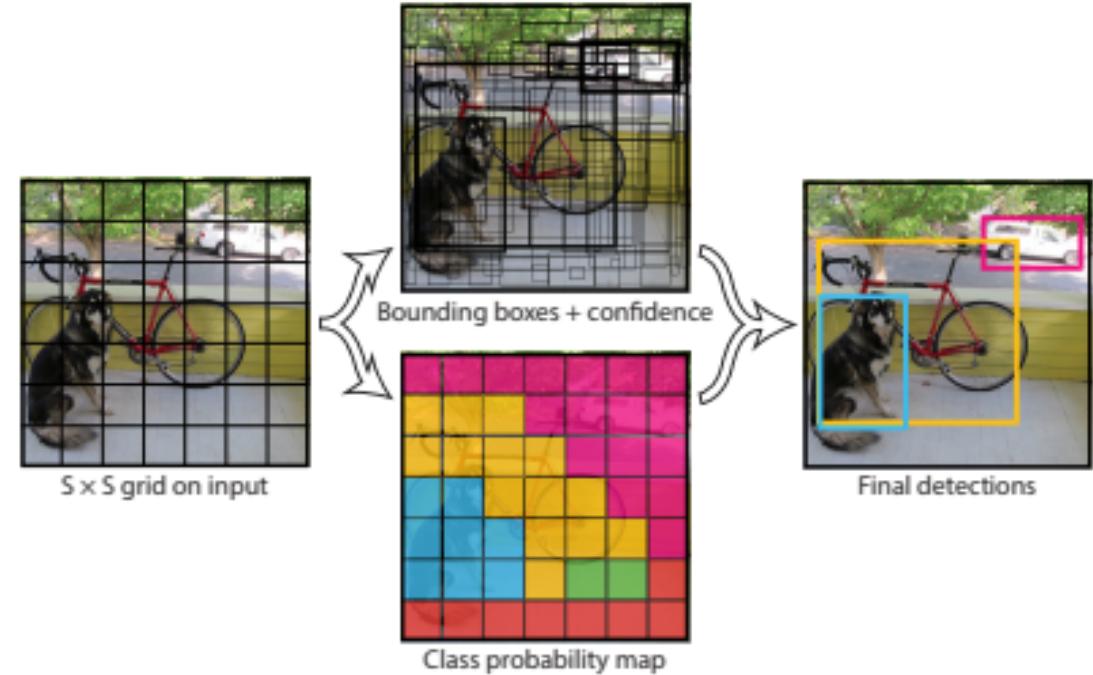
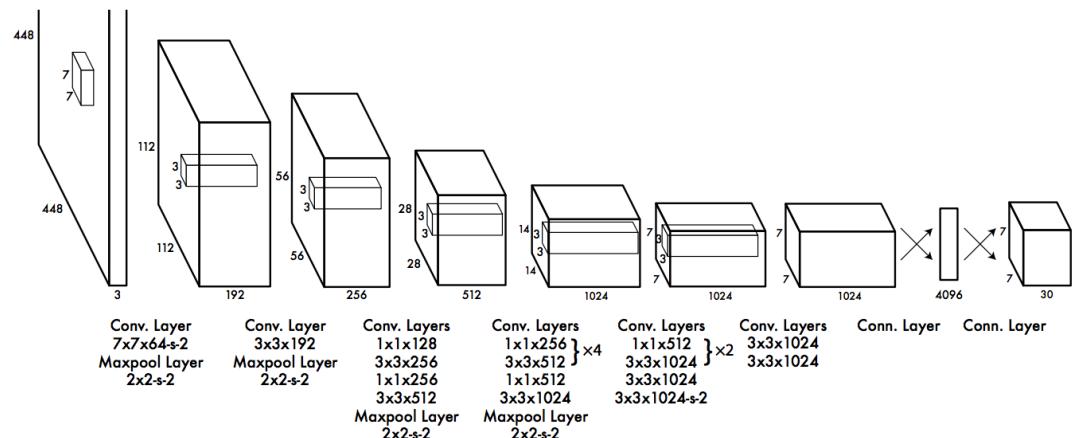


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

引用元： You only look once: Unified, real-time object detection, edmon, Joseph, et al,
<https://arxiv.org/pdf/1506.02640.pdf>

YOLO

- ・グリッドサイズはユーザーが決める。(原論文では 7×7 で検証を実施)
- ・グリッド内で出力できるクラスは1つのみ。
- ・グリッド内で検出できる物体の数はユーザーが決める。(原論文では2で検証を実施)
- ・上記の理由により、グリッド内に大量のオブジェクトが写ってしまうような場合は精度が低くなる。
- ・YOLOのアーキテクチャを以下に示す。



引用元： You only look once: Unified, real-time object detection, edmon, Joseph, et al, <https://arxiv.org/pdf/1506.02640.pdf>

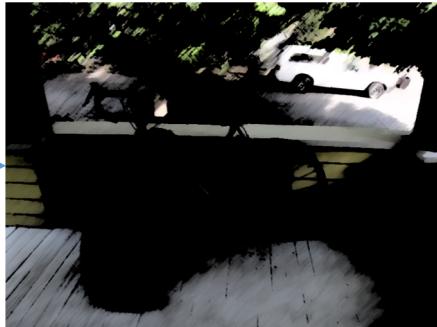
Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

YOLOの仕組み

- ・ 入力画像をCNNによって、特徴マップに変換する。
- ・ 特徴マップ全体を見渡して、各セル毎に、中心が存在しているような物体とそのクラスを同時に予測する。

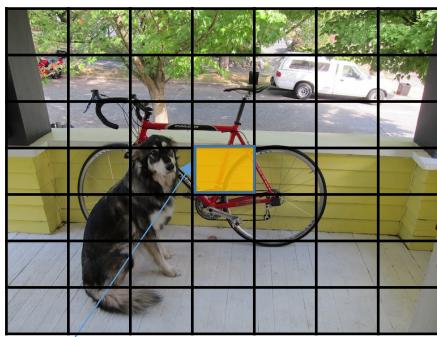


CNN

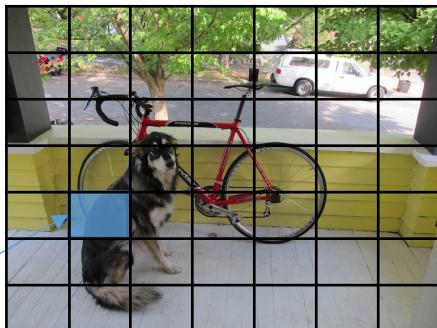


特徴マップ

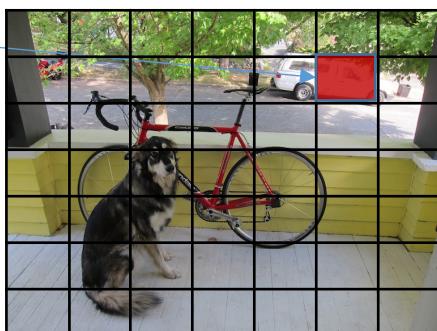
全結合
全結合
全結合



「特徴マップ全体を見渡す限り、このセルには、**自転車**という物体の中心が存在しているそうだな」



「特徴マップ全体を見渡す限り、このセルには、**犬**という物体の中心が存在しているそうだな」



「特徴マップ全体を見渡す限り、このセルには、**自動車**という物体の中心が存在しているそうだな」

- SSD(single shot multibox detector)は、YOLOと似たアルゴリズム。
- 様々な階層の出力層からマルチスケールな検出枠を出力できるように工夫されている。
- CNNでは、出力層に近いほど特徴マップのサイズが小さくなるので、それぞれの層の特徴マップには様々なサイズの物体を検出できる情報があるはずである。つまり、出力層に近い特徴マップほど大きな物体を検知しやすくなり、入力層に近い特徴マップほど小さな物体を検知しやすくなると考えられる。
- YOLOより高速で、Faster R-CNNと同等の精度。

複数のスケールの特徴マップを用いて、クラス分類問題とバウンディングボックス回帰問題を解いている。

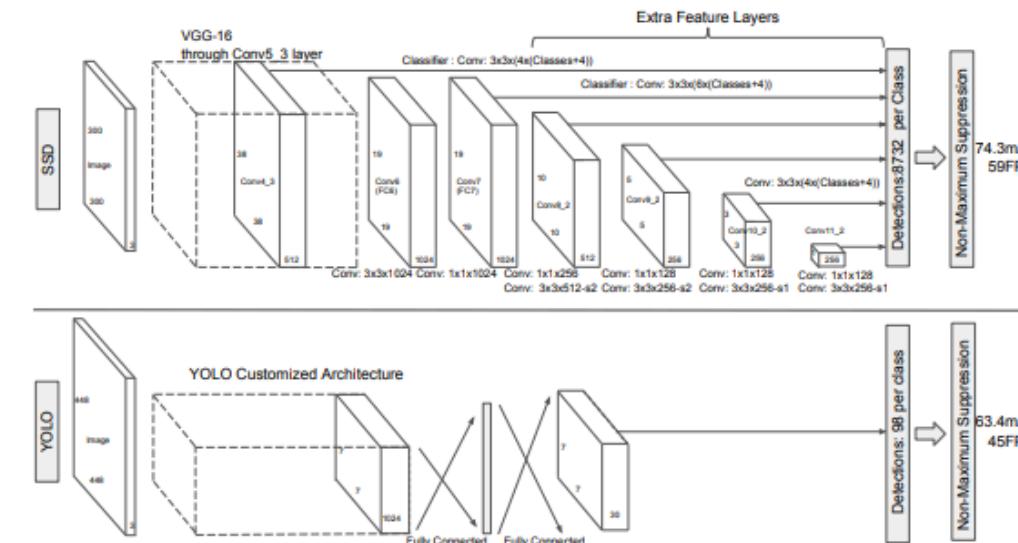


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300×300 input size significantly outperforms its 448×448 YOLO counterpart in accuracy on VOC2007 test while also improving the speed.

引用元： SSD: Single Shot MultiBox Detector, Liu, Wei, et al.
<https://arxiv.org/pdf/1512.02325.pdf>

物体検出方法の比較

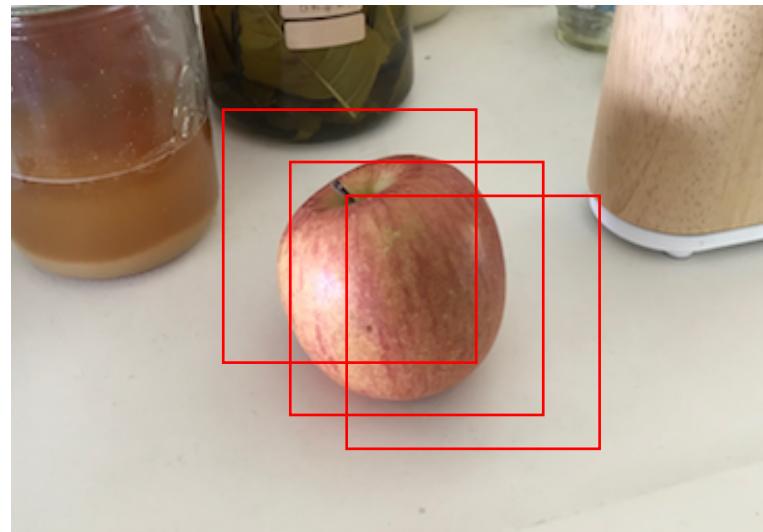
※BB : バウンディングボックス

※特徴マップとは、CNNで計算された特徴量のこと

	R-CNN	Fast R-CNN	Faster R-CNN	YOLO	SSD
特徴	約2000の物体領域候補のそれぞれについて、特徴量を算出する必要がある。	画像全体を入力して計算した特徴マップをすべての提案された領域で再利用することで、R-CNNを高速化させた。	特徴マップから物体領域候補を提案するRPNとFast R-CNNの2つのネットワークで構成される。	画像をグリッド状に分割して考えるのが特徴。分割された1つの領域をセルと呼ぶ。1つのNNで、クラス分類問題とBB回帰問題を解く。	複数のスケールの特徴マップを用いる。
BBの決定方法	選択的検索法で提案された物体領域候補ごとに、特徴マップとBBを回帰させる問題を解く。		RPNで提案された物体領域候補ごとに、特徴マップとBBを回帰させる問題を解く。	特徴マップとBBを回帰させる問題を解く。	
クラス分類の方法	特徴マップを線形SVMに入力しクラスを予測する。	特徴マップを用いて、ニューラルネットワークによりクラス分類問題を解く			
サイズを揃えるための工夫	異なる物体領域候補のサイズを一定のサイズに変換してから、CNNに入力する。	RoIPooling層を利用し、サイズの異なる特徴マップを一定の大きさのベクトルに変換する		特徴マップは一つであるため、サイズを揃える処理は不要。ただし、NNに入力される画像サイズを揃える処理は必要。	

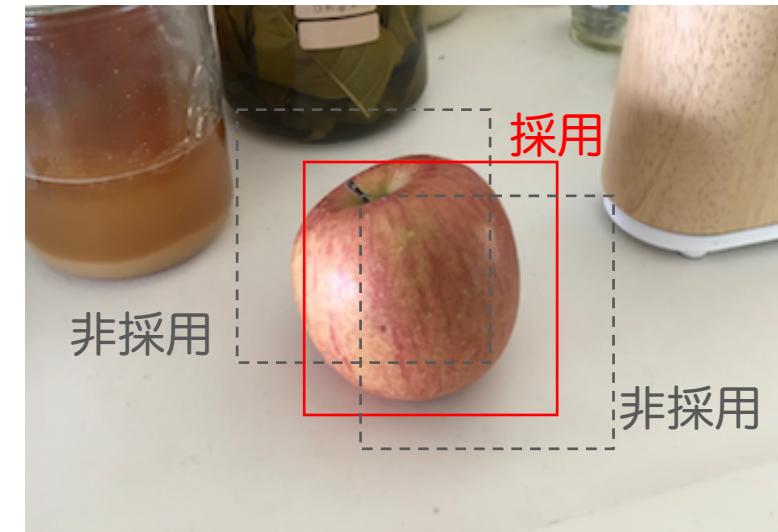
非最大値の抑制

- 物体検出の結果として、同一物体に対して複数のバウンディングボックスが検出されてしまうことがある。
- これを防ぐ方法として、非最大値抑制(non-maximal suppression, NMS)がある。



同一の物体に対して、複数のバウンディングボックスが検出されている

NMS
→



NMSによって、バウンディングボックスを1つに絞ることができる

非最大値の抑制

- 非最大値抑制の手順

1. スコア(信頼度など)が最も高いバウンディングボックス(以下、BB)に「採用」とマークし、そのBBと一定の割合以上の重なりをもつ(IoUが閾値以上)領域に「非採用」とマークする。
2. 「採用」もしくは「非採用」のマークが付いていないBBの中で、スコアの最も高いBBに「採用」とマークし、これと一定の割合以上の重なりをもつ(IoUが閾値以上)BBに「非採用」とマークする。
3. 全てのBBについて、「採用」または「非採用」のいずれかがマークされるまで上記の手順を繰り返す。
4. 「採用」とマークされたBBの集合が最終的な検出結果となる。

物体検出の評価指標

- ・ バウンディングボックスの一致度を測る指標
 - ・ IoU
- ・ クラスの一致度を測る方法
 - ・ AUC(area under curve)
 - ・ ROC曲線の下部面積
 - ・ AP(average precision)
 - ・ PR(precision recall)曲線の下部面積
 - ・ mAP(mean average precision)
 - ・ APの全クラス平均

<クラスの一致度を測る方法について>

- ・ 物体検知のコンテストでは、独自の定義がされる場合があるので、具体的な計算方法はそれを参照するのがよい。
- ・ 例えば、有名な物体検出コンテストの一つであるThe PASCAL Visual Object Classes (VOC) Challengeでは、年によって計算方法が変わっている。

2006年以前 : ROC曲線に基づくAUC

2007年~2009年 : AP(11点の近似計算)

2010年~2012年 : AP(近似をせずに詳細に計算)

<http://host.robots.ox.ac.uk/pascal/VOC/>

物体検出の評価指標, IoU

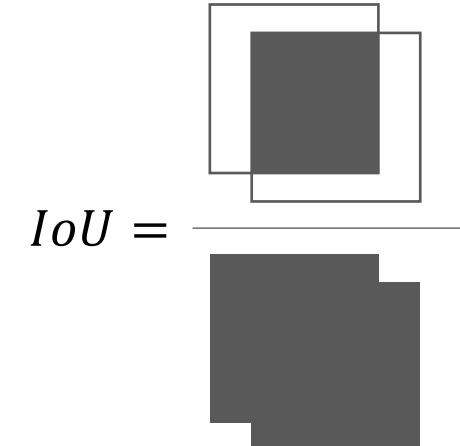
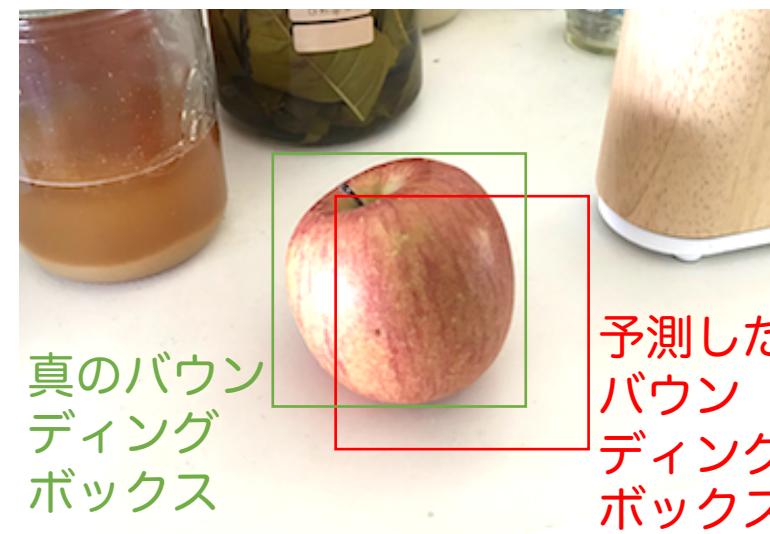
- 予測したバウンディングボックス \mathcal{R}_p と真のバウンディングボックス \mathcal{R}_g の一致度を定量化する指標として、IoU(intersection over union)がある。
- 一般に、IoUが0.5を超える場合に、良いバウンディングボックスの予測結果であると判断されることが多い。
- IoUの定義

$$IoU = \frac{area(\mathcal{R}_p \cap \mathcal{R}_g)}{area(\mathcal{R}_p \cup \mathcal{R}_g)}$$

area() : 指定した領域の面積

\mathcal{R}_p : 予測したバウンディングボックス

\mathcal{R}_g : 真のバウンディングボックス



物体検出の評価指標, APとmAP

- クラスの一致度の評価には、クラスごとの平均適合率(average precision, AP)が利用されることが多い。
- 平均適合率を全クラス平均したものは、mAP(mean average precision)と呼ばれる。

$$\text{mAP} = \frac{1}{N_c} \sum_k^{N_c} AP_k$$

AP_k : クラス k の平均適合率

N_c : クラスの数

物体検出の評価指標, APの計算方法

- APにはいくつかの算出方法があるが、ここでは、2007年~2009年のThe PASCAL Visual Object Classes (VOC) Challengeで用いられた方法を紹介する。
- 参照論文
 - The PASCAL Visual Object Classes (VOC) Challenge,
<http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf>
- 参考ブログ
 - mAP (mean Average Precision) for Object Detection,
https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

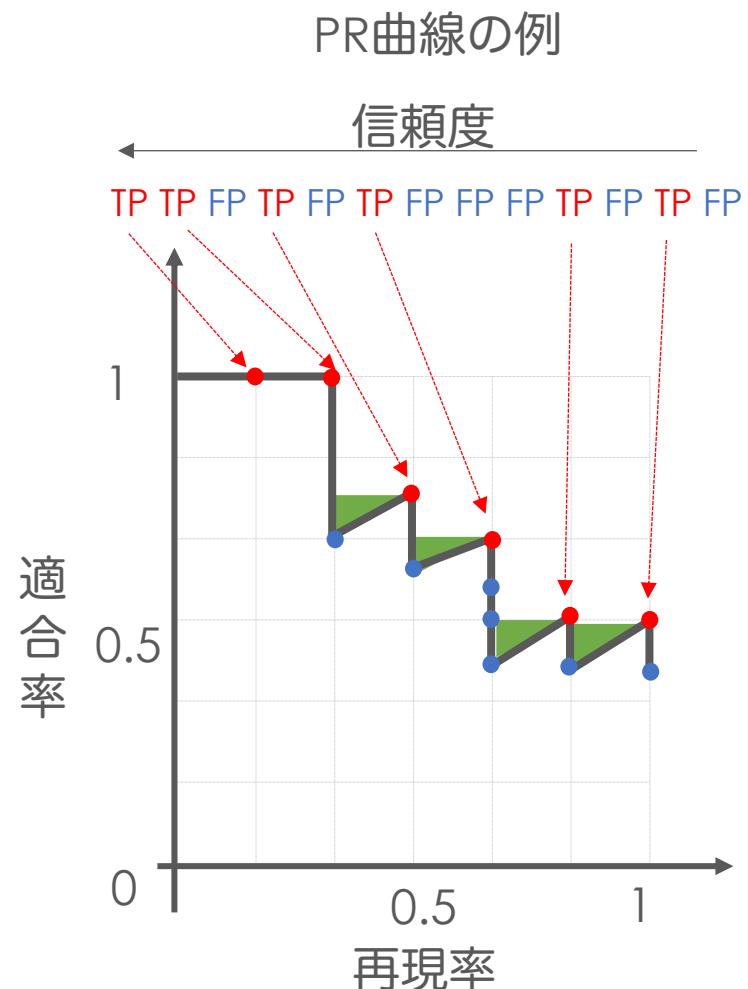
物体検出の評価指標, APの計算方法

- APの計算方法(VOC2007版)

- ここでは、正解のバウンディングボックスを物体領域、検出されたバウンディングボックスを検出領域と呼ぶことにする。
- 物体検出結果として、検出領域とその領域の信頼度のスコアを算出する。
- 各検出領域について、物体領域とのIoUを算出する。
- IoUが50%を超え、かつIoUが最も高い検出領域を物体領域に対応させる。
- 物体領域に対応づけられた検出領域は真陽性(true positive, TP)として扱い、それ以外の検出領域は偽陽性(false positive, FP)として扱う。
- 検出領域を信頼度の降順で並べ替え、1つずつ検出領域を追加しながら、適合率(精度とも呼ばれる)と再現率を算出する。
- 最後まで計算できたら、全ての適合率と再現率のペアをプロットする。これをPR(precision recall)曲線という。
- 以下の方法でPR曲線の下部面積を近似的に算出する。この算出結果をAPと呼ぶ。

$$p_{interp}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r})$$

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.01, 0.02, \dots, 1\}} p_{interp}(r)$$



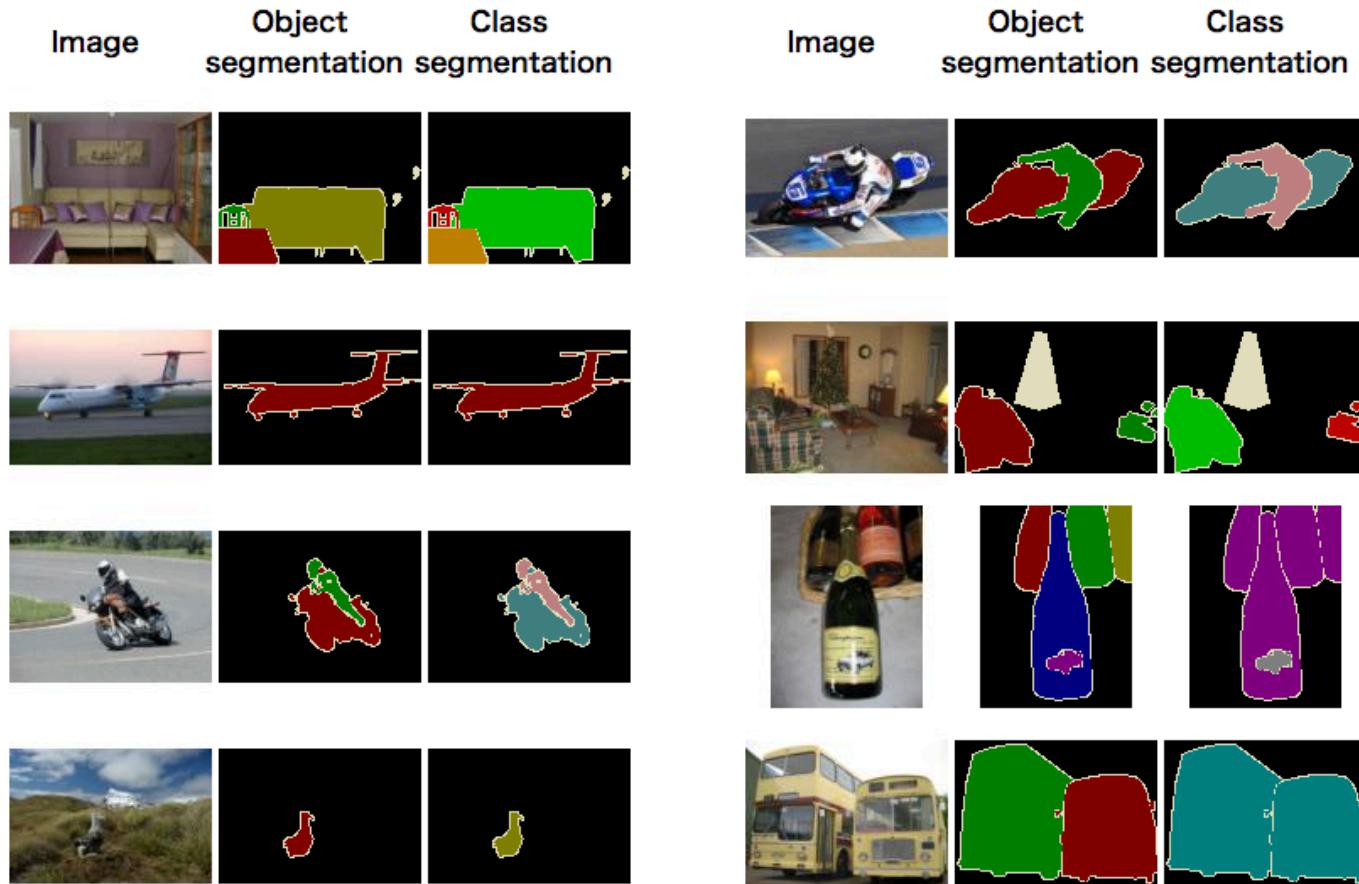
緑部の面積は、近似的に面積を算出したことによって補間された部分

Any Questions?

セマンティックセグメンテーションタスクとCNN

セマンティックセグメンテーションタスク

- セマンティックセグメンテーションタスクとは、画像を画素レベルで把握するために各画素に対してオブジェクトクラスを割り当てるタスク。簡単に言うと、ピクセル単位の分類問題。
- 自動運転や医療画像の分野において重要な技術。
- 画像分類同様、セマンティックセグメンテーションにおいても CNNは多くの成果を収めている。



セマンティックセグメンテーションの例

引用元：<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/segexamples/index.html>

- FCN(Fully Convolutional Network, 全層畳み込みネットワーク)は、セマンティックセグメンテーションのためのネットワーク。全結合層を使わないCNN。
- CNNの全結合層を畳み込み層に置き換えることで、出力を分類クラスではなく二次元マップに変えることができる。
- FCN以前はネットワーク内に全結合層が存在することにより、固定サイズの画像しか扱うことができなかった。FCNでは、全結合層が存在しないため、あらゆるサイズの画像でセグメンテーションマップが生成できる。

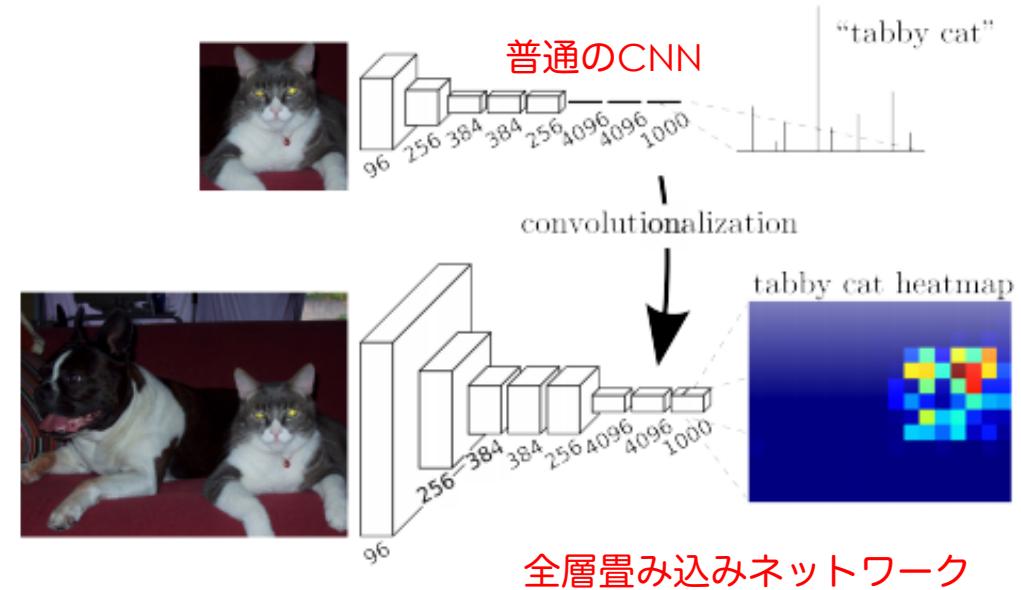


Fig. 2. Transforming fully connected layers into convolution layers enables a classification net to output a spatial map. Adding differentiable interpolation layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end pixelwise learning.

引用元：Fully Convolutional Networks for Semantic Segmentation,
Evan Shelhamer, Jonathan Long, Trevor Darrell,
<https://arxiv.org/pdf/1605.06211.pdf>

- プーリング層では、特徴マップはダウンサンプリングされる。
- FCNでは、プーリング層の後、**逆畳み込み**を用いて**アップサンプリング**を施すことで、出力される画像サイズを大きくする。
- 損失は、ピクセル毎のクロスエントロピー誤差の合計。

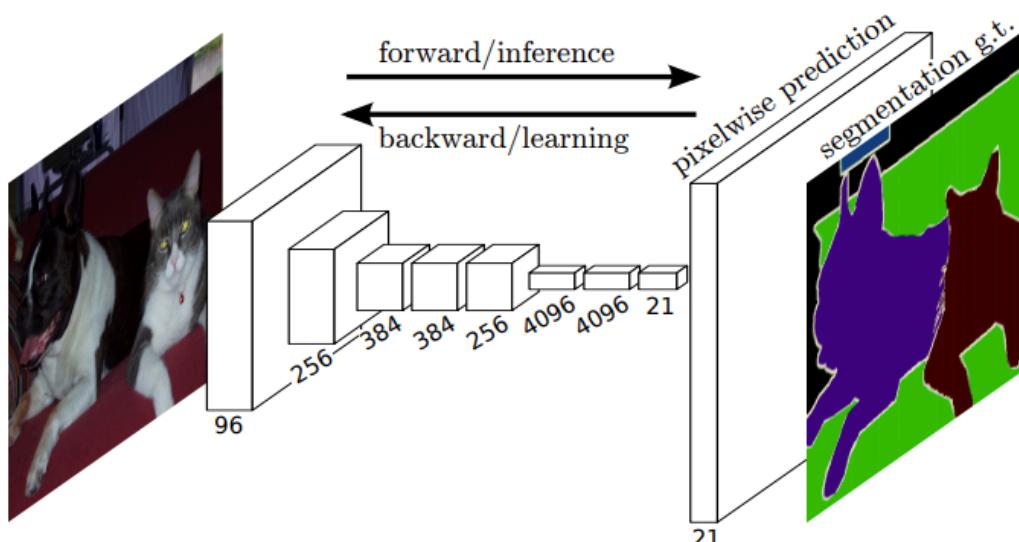
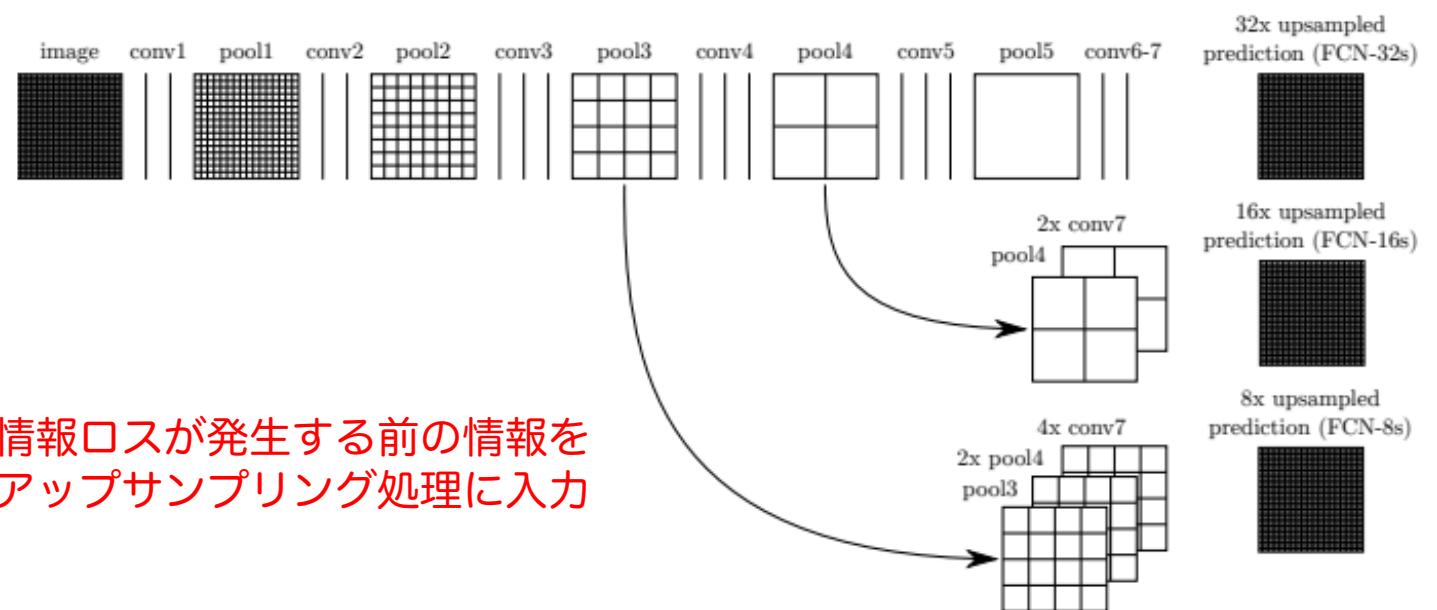


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

引用元：Fully Convolutional Networks for Semantic Segmentation,
Evan Shelhamer, Jonathan Long, Trevor Darrell,
<https://arxiv.org/pdf/1605.06211.pdf>

- ・ ダウンサンプリングされた特徴マップに対して単純に逆畳み込みを適用するだけでは、出力結果が粗くなってしまうため、**スキップ接続**を用いて**情報ロスが発生する前の情報をアップサンプリング処理に入力する。**



引用元：Fully Convolutional Networks for Semantic Segmentation, Evan Shelhamer, Jonathan Long, Trevor Darrell, <https://arxiv.org/pdf/1605.06211.pdf>

Fig. 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the pool4 layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from pool3, at stride 8, provide further precision.

SegNet

- エンコーダ(encoder)部分とデコーダ(decoder)部分で構成されるセマンティックセグメンテーションのためのネットワーク。
- 損失は、ピクセル毎のクロスエントロピー誤差の合計。

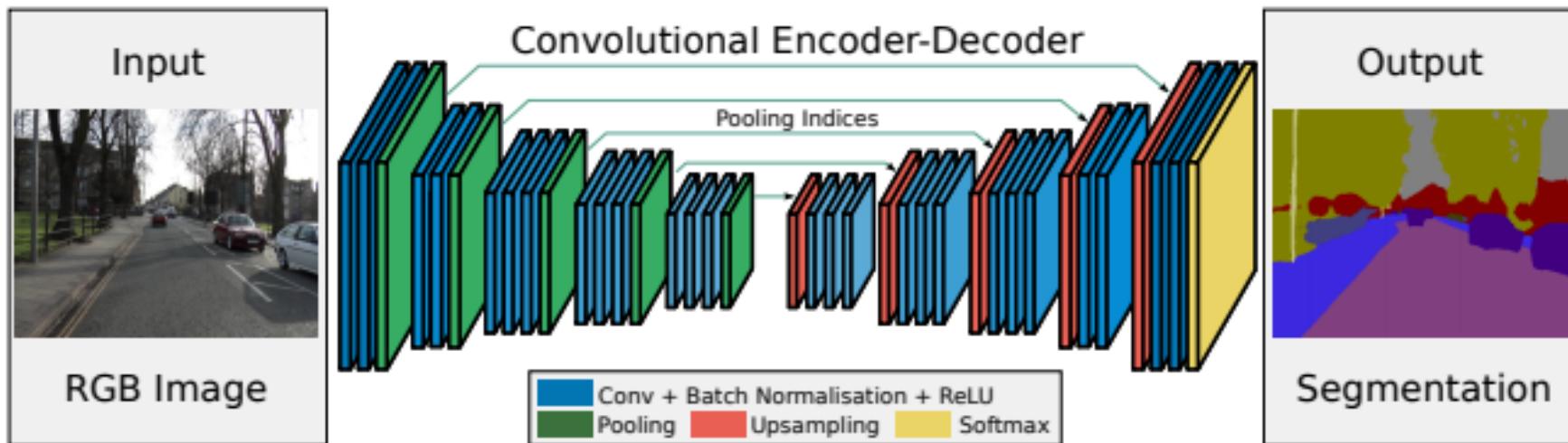
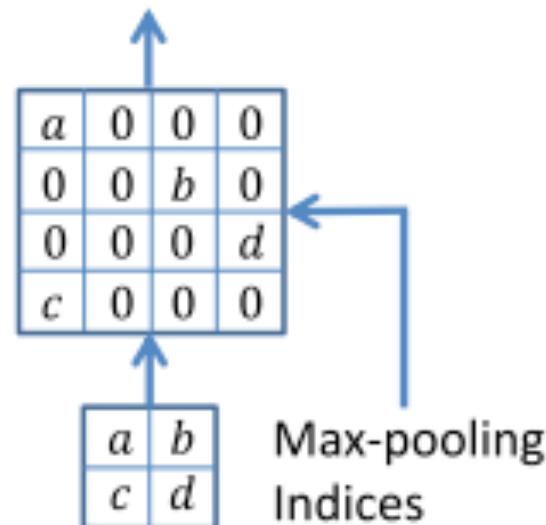


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

引用元 : SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, Vijay Badrinarayanan, et al.
<https://arxiv.org/pdf/1511.00561.pdf>

SegNet

- FCNのようにプーリング前の特徴をアップサンプリング層にコピーするのではなく、エンコーダ部分のマックスプーリング層で採用した値の場所を記録しておき、デコーダ部分のアップサンプリング時にその場所を使って特徴マップを拡大する。
- これにより位置情報が保持される。また、FCNに比べてメモリの効率が良くなつた。

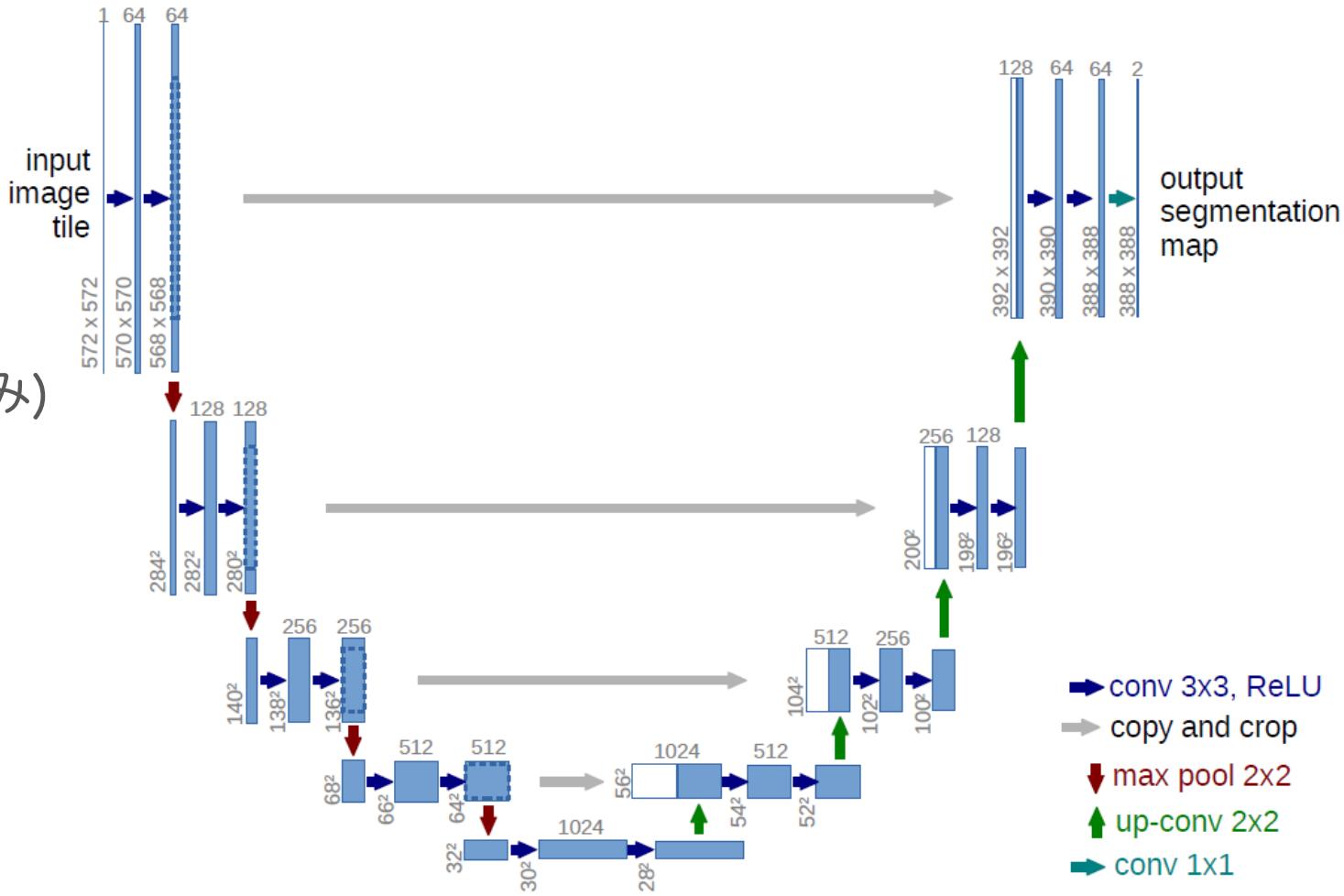


U-Net

- セマンティックセグメンテーションタスク用のネットワーク
- 形がU字型をしているからU-Net
- Fully Convolutional Network(FCN), Segmentation Network(SegNet)を破って SotA(その時点で最高の性能)
- 多くの派生系があり、特に医用画像のセグメンテーションでは主流
- 原論文
 - U-Net: Convolutional Networks for Biomedical Image Segmentation, Olaf Ronneberger, Philipp Fischer, Thomas Brox,(MICCAI 2015)
 - 医用画像などデータ数が少ない場合でもうまく動くセグメンテーション用CNNを提案

U-Net

- ・ 見た目通りのU-Net
- ・ 横向きのパスは畳み込み
- ・ 下向きのパスはmax-pooling
- ・ 上向きのパスはup-conv (逆畳み込み)
- ・ 最終的な特徴マップを 1×1 convでセグメンテーションマップに変換

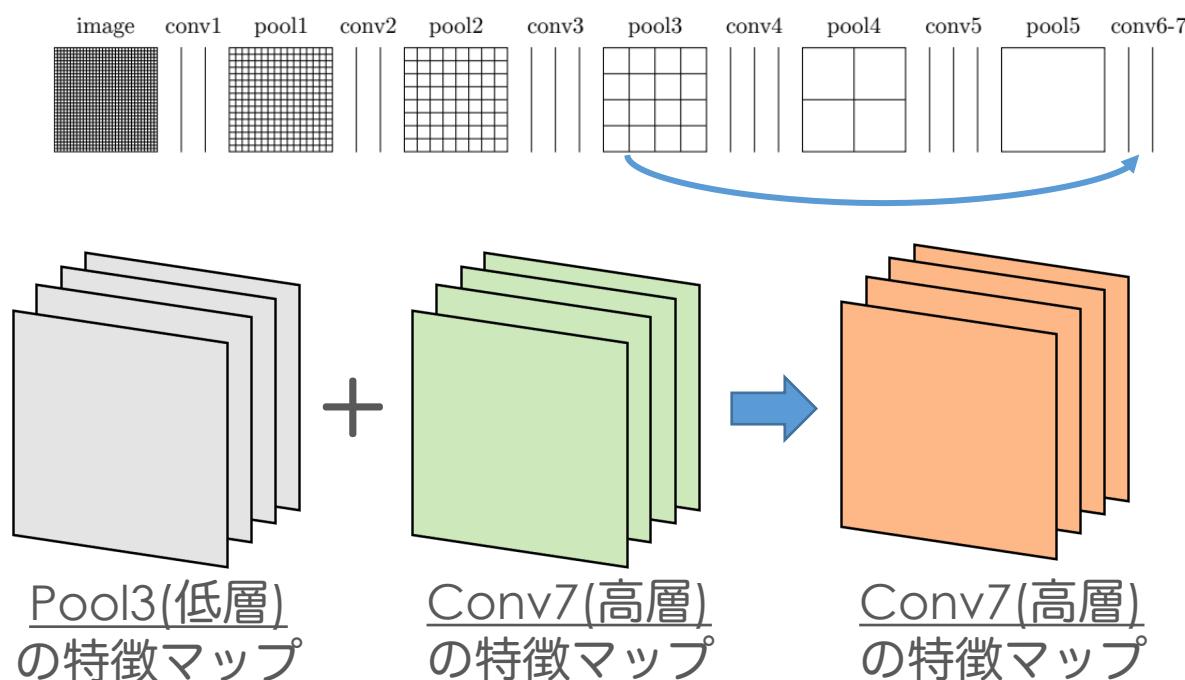


(原論文 Fig. 1から引用)

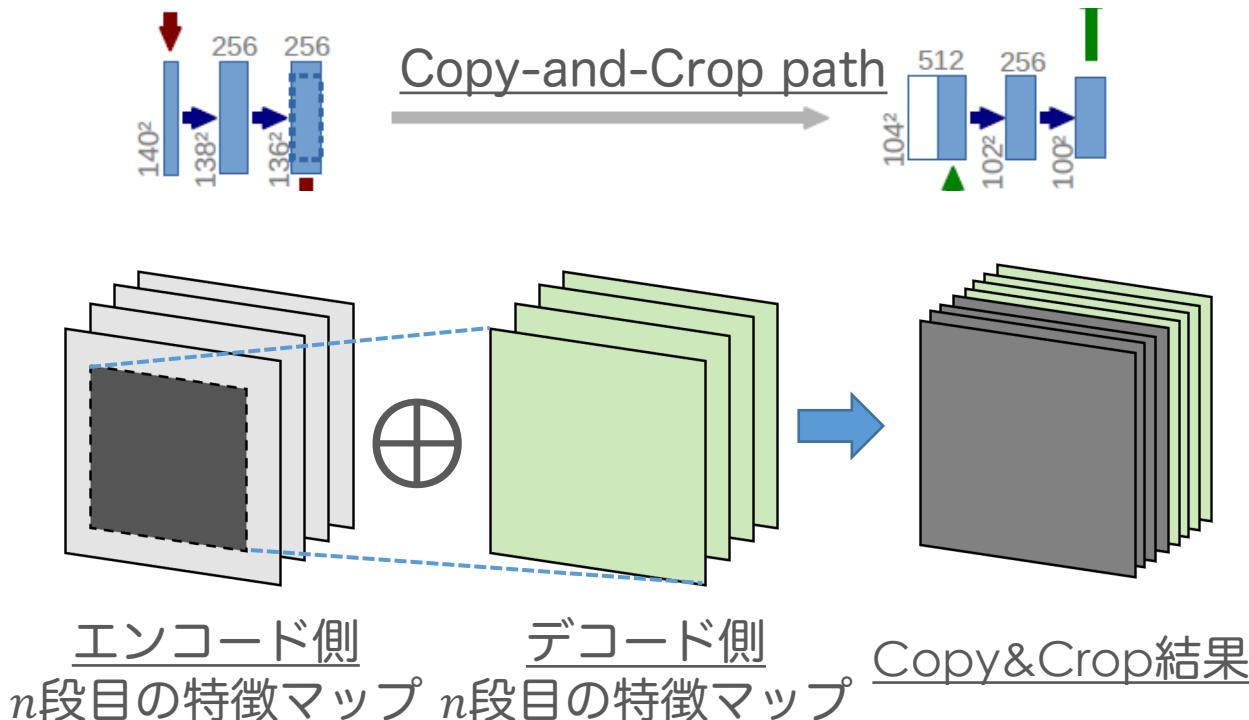
U-Net

- FCNでは入力側の中間表現を出力側に反映する際、チャンネルごとの足し算を行っていた
- U-Netでは足し算ではなく”連結”することで入力を位置情報を保持している

FCNにおけるSkip-connection



U-NetにおけるCopy-and-Crop



Any Questions?

[グループワーク] 置み込み演算の調査

E資格対策

- ・近年、様々な置み込みの方法が提案されています。
- ・以下に挙げる置み込み方法を調べてみましょう。
 - Depthwise convolution
 - Pointwise convolution
 - Grouped convolution
 - Dilated convolution
 - Transposed convolution
- ・参考サイト
 - https://github.com/vdumoulin/conv_arithmetic
 - <https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>
 - <https://www.youtube.com/watch?v=T7o3xvJLuHk&feature=youtu.be>
 - <https://arxiv.org/pdf/1603.07285v1.pdf>
 - <https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d>
 - <https://www.youtube.com/watch?v=T7o3xvJLuHk&feature=youtu.be&app=desktop>
- ・2~3名のグループに分かれて、上記課題に取り組みましょう。(45分)
- ・最後に、グループごとに発表していただきます。(15分)

MoblieNet

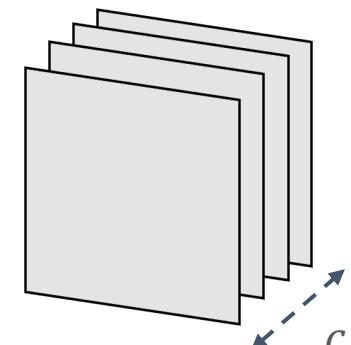
MobileNet

- 論文
 - MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, A. G. Howard et al. ,(Arxiv preprint, from Google)
- 主な提案
 - Depthwise Separable ConvolutionによるCNNの計算時間の削減手法
 - ハイパーパラメータで計算時間と認識性能をうまくハンドリングする手法
- 提案の背景
 - 従来のCNNモデルは高性能であるが計算が非常に重い。
 - CNNを軽量化し、スマホなどのモバイル機器でも使えるようにしたい。
 - リアルタイム認識の場合にレイテンシ(≒計算時間)をうまくコントロールしたい。

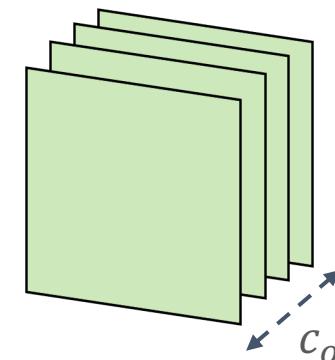
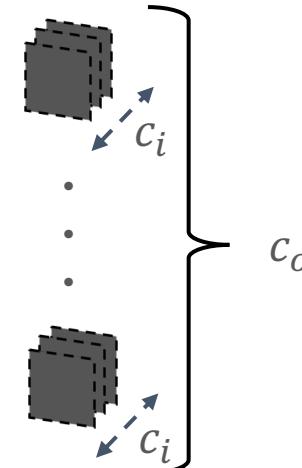
畳み込み処理のパラメータ数

- ・ 従来のConvolution処理の何が重い？
- ・ 入力のチャンネル数 c_i , 出力のチャンネル数 c_o , 畳み込みカーネルのサイズ k^2 のときの畳み込み層の学習パラメータ数 $N_p = c_i \cdot k^2 \cdot c_o$
 - ・ Convolutionの”計算量”は入力が $h \times w$ ならば $O = h \cdot w \cdot N_p$
 - ・ パラメータ数を減らすことは当然計算量を減らすことに直結
※ (計算時間には直結しないことに注意は必要)
- ・ 従来の畳み込みでは 入力マップの”チャンネル間”的相関を律儀に計算している部分がボトルネックになりうる ($\times c_i$)
 - ・ ⇒ この部分を効率化すれば軽量なConvolutionになるのでは？

入力マップ

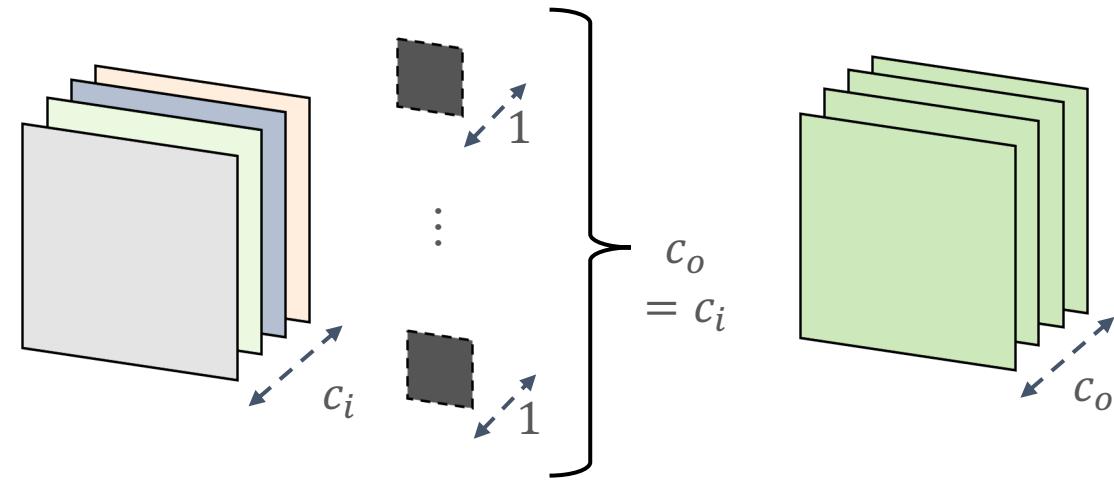


畳み込みフィルタ



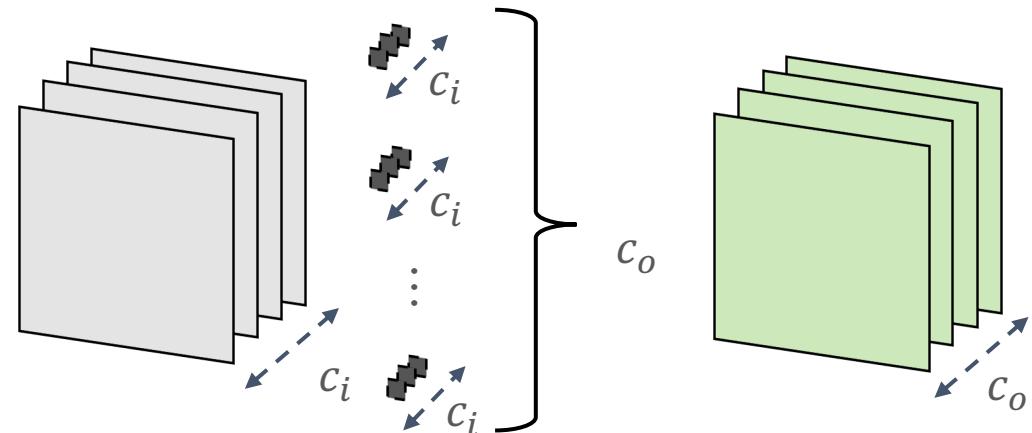
Depthwise Convolution

- チャンネルごとに空間的畳み込みを行う。
- c 間の相関は取らない。
 - $N_p = c_o \cdot k^2$



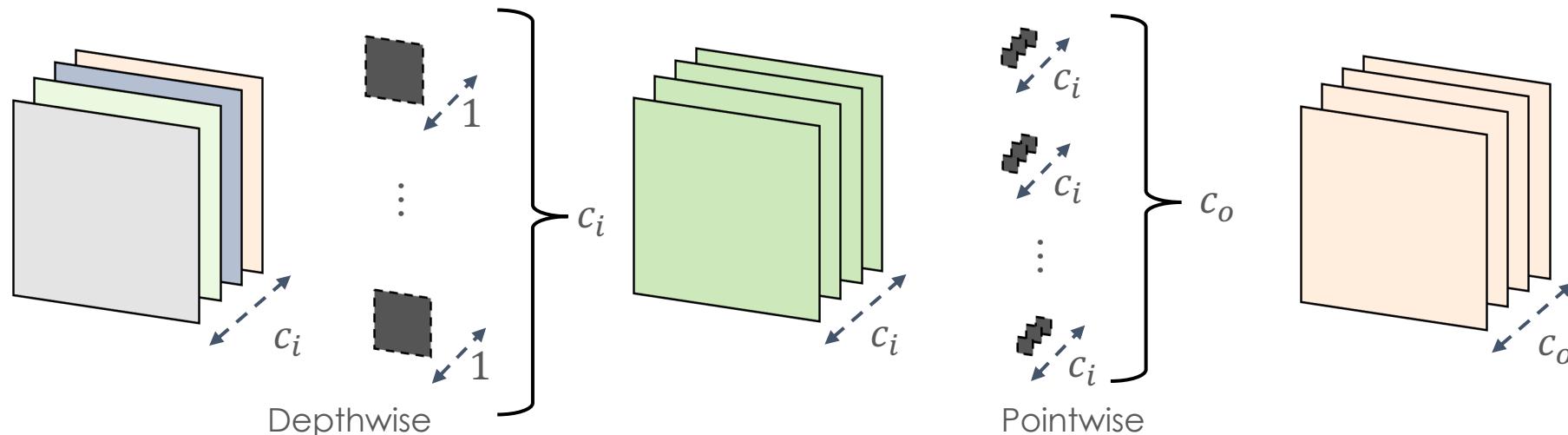
Pointwise Convolution

- $N_p = c_i \cdot k^2 \cdot c_o$ で k^2 を小さくする方策がある
 - $k = k^2 = 1$ ならば $N_p = c_i \cdot c_o$
- 空間的特徴(形状)を捉えずにチャンネル間の相関構造だけを変える特殊な畳み込み演算
- Pointwise Convolution, 1x1 Convolution, Cascaded Cross Channel Pooling (CCCP) など色々な名前がある
- 非常にわかりにくいが、**モダンなCNNでは超重要技術**
 - チャンネル数の辻褄を合わせることができる



Depthwise Separable Convolution

- Depthwise Convolutionは非常に軽量だが、チャンネル間の構造を捉えることができない
 - 出力のチャンネル数 $c_o = c_i$ に固定されてしまうし、性能が出ない
- Pointwise Convolutionと継続接続することで「あとで」チャンネルの処理を行う



- パラメータ数は $N_p = c_i(c_o + k^2)$
 - ありがちな $k = 3$; $c_i, c_o > 64$ の場合ではパラメータ数はおよそ1/9に減少する

[宿題] 論文調査

- ・ 深層学習は、日々新しい手法が開発されている分野です。
- ・ 最新の手法は書籍に掲載されていないことが多く、その場合は原論文を読む必要があります。ほとんどの論文は、 Arxiv.org(<https://arxiv.org>)に登録されています。
- ・ Arxiv.orgのサイトから、通し課題に役立ちそうな論文を探してみましょう。
- ・ 探す時のキーワード
 - ・ CNN
 - ・ Convolutional
 - ・ Recognition
 - ・ Neural network
 - ・ MNIST
 - ・ など

Any Questions?

その他の話題

目次

1. 構造出力
2. CNNで扱うデータの種類
3. 転移学習

構造出力

- ・ 置み込みニューラルネットワークでは、クラスラベルや実数値を出力するだけでなく、高次元な構造を持つオブジェクトを出力することもできる。
- ・ 例えば、入力画像の各ピクセルに対応するラベルを予測するタスクなどに利用できる。これは、パッチ分類と呼ばれ、セマンティックセグメンテーションに利用される。
 - ・ 「セマンティックセグメンテーションタスクとCNN」の節を参照されたい。
- ・ 参考：『深層学習, Ian Goodfellow, KADOKAWA, p.257』

CNNで扱うデータの種類

CNNで扱うデータの種類

- CNNで扱えるデータは画像だけではない。
- 扱えるデータの例を以下に示す。
- 参考：『深層学習, Goodfellow, KADOKAWA, p.258』

	単一チャンネル	複数チャンネル
1次元	時系列データ (時間の1次元で、何らかの値が1チャンネル。音声データなど。畠み込みを行う軸は時間軸)	スケルトンアニメーションデータ (時間の1次元で、ある関節のある軸が1チャンネルになりそれが複数ある)
2次元	グレースケール画像データ (縦と横の2次元で、グレースケールの1チャネル) フーリエ変換で事前処理された時系列データ (周波数軸と時間軸の2次元で、スペクトルが1チャンネル)	カラー画像データ (縦と横の2次元で、RGBの3チャネル)
3次元	体積データ (縦と横と奥行きの3次元で、何らかの値が1チャンネル。CTスキャンのデータなど)	カラー動画データ (縦と横と時間の3次元で、RGBの3チャネル)

転移学習

- ・ 転移学習(transfer learning)とは、ある問題設定で学んだことを別の問題設定の汎化性能向上に役立てること。
- ・ 異なる問題設定間において、共通の特徴量を仮定できるとき、転移学習は有効である。
- ・ 参考：『深層学習, Ian Goodfellow, KADOKAWA, p.392』
- ・ 転移学習の例
 - ・ ある問題設定で学習させた特徴量を他の問題設定で利用する学習方法
 - ・ ワンショット学習
 - ・ ゼロショット学習

<参考>

転移学習に似た言葉として再学習(fine tuning)という言葉がある。再学習という言葉の意味は、明確に定まっていないが、以下のようなことを指すことが多い。

- ・ 学習済みモデルのパラメータのうち入力層から出力層の数層手前までのパラメータをそのまま利用し、これに新しい層を結合して、その新しい層のパラメータを更新すること。
- ・ 教師なし学習により事前学習済みのパラメータを初期値にして、教師あり学習によってそのパラメータを更新すること。

ワンショット学習

- ・ 転移学習の極端な形式として、ワンショット学習 (one shot learning)がある。
- ・ 転移先のタスクにおいてはラベル付き事例が1つだけ与えられる。
- ・ ワンショット学習の事例
 - ・ Siamese Neural Networks for One-shot Image Recognition, Gregory Koch, Richard Zemel , Ruslan Salakhutdinov,
 - ・ <https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf>

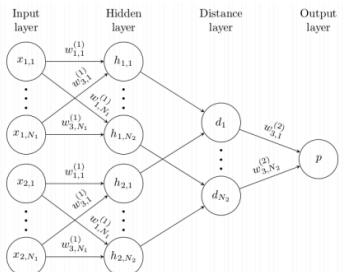


Figure 3. A simple 2 hidden layer siamese network for binary classification with logistic prediction p . The structure of the network is replicated across the top and bottom sections to form twin networks, with shared weight matrices at each layer.

- <この論文で提案された手法の概要>
- ・ 2つの画像が同じクラスであるかを予測するモデルを構築しておく。
 - ・ これに、新しいクラスの画像(下記の例ではバスケットボール)を1枚だけ学習させる。
 - ・ すると、次からバスケットボール画像のクラス分類ができるようになる。

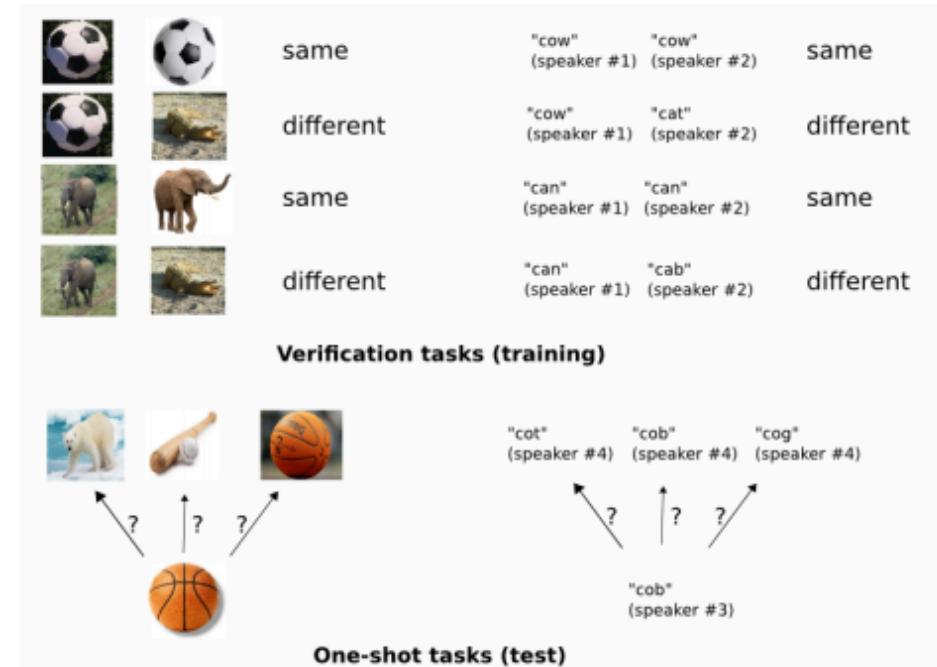


Figure 2. Our general strategy. 1) Train a model to discriminate between a collection of same/different pairs. 2) Generalize to evaluate new categories based on learned feature mappings for verification.

ゼロショット学習

- 転移学習の極端な形式として、ゼロショット学習(zero shot learning)がある。
- 転移先のタスクにおいてはラベル付き事例が与えられない。
- ゼロショット学習の事例
 - Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat
 - 参考ブログ：<https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html>

Google Neural Machine Translation
(GNMT)を用いた事例

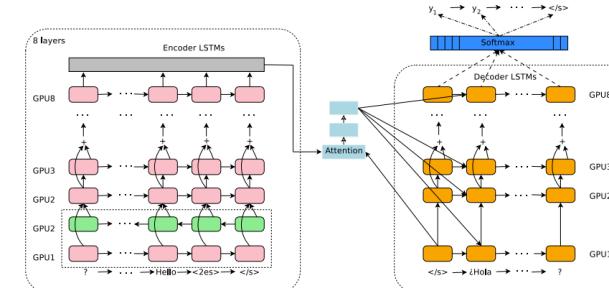
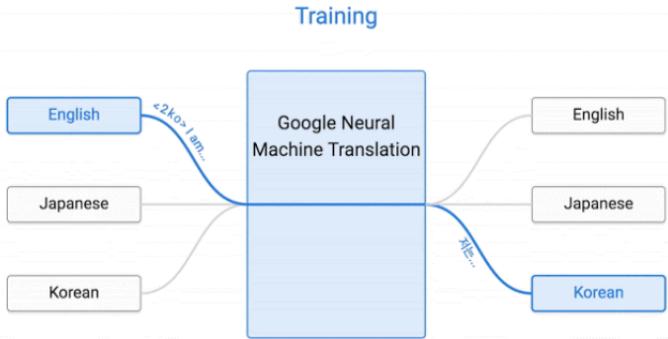


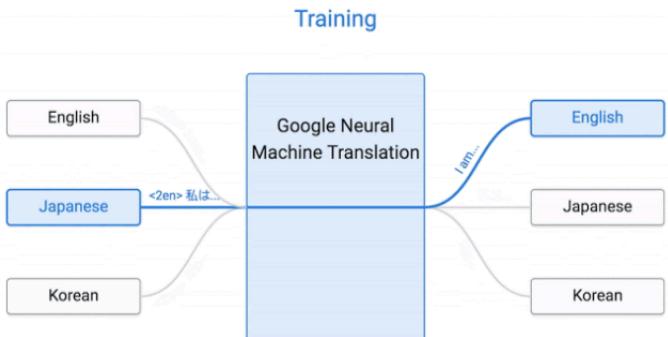
Figure 1: The model architecture of the Multilingual GNMT system. In addition to what is described in [24], our input has an artificial token to indicate the required target language. In this example, the token "<2es>" indicates that the target sentence is in Spanish, and the source sentence is reversed as a processing step. For most of our experiments we also used direct connections between the encoder and decoder although we later found out that the effect of these connections is negligible (however, once you train with those they have to be present for inference as well). The rest of the model architecture is the same as in [24].

ゼロショット学習

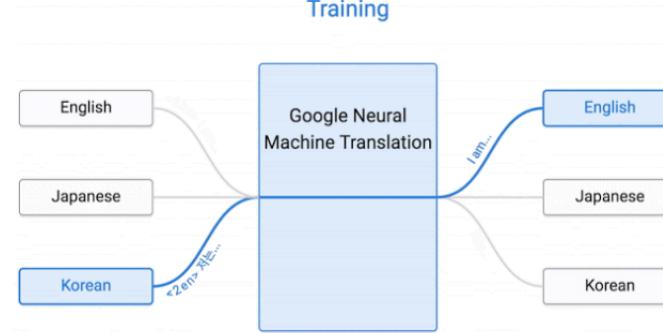
英語->韓国語を学習



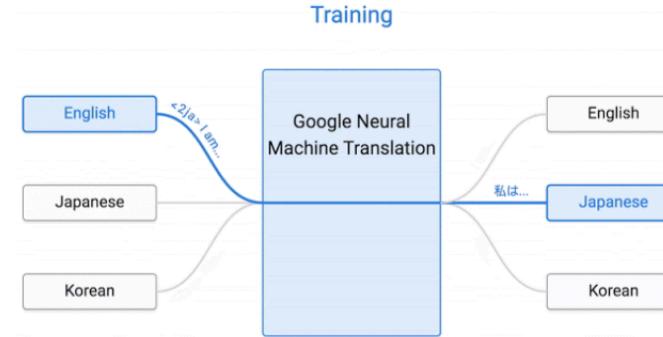
日本語->英語を学習



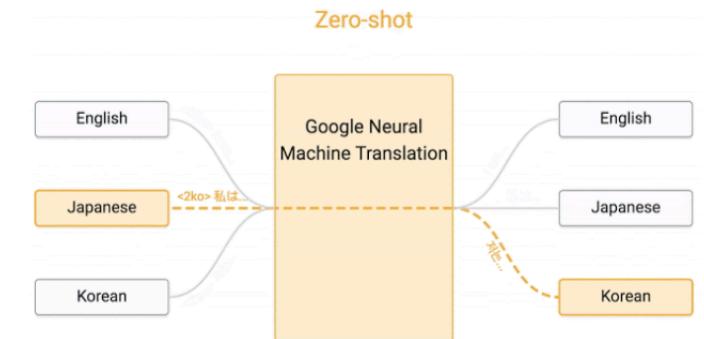
韓国語->英語を学習



英語->日本語を学習



学習したことがない
日本語->韓国語も翻訳できる
ようになった



原著論文では、中間言語のような
ものを獲得できたと表現されて
いる

Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat

Any Questions?

講座の時間が余ったら

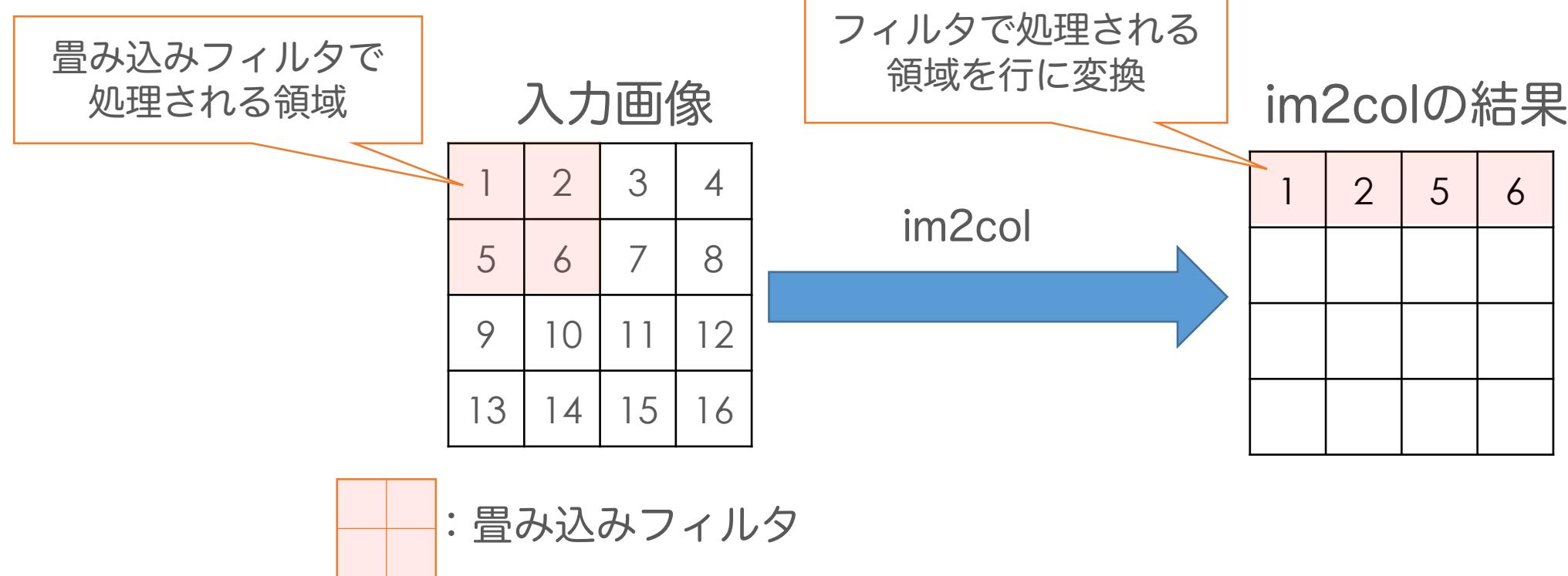
- ・今回の復習をします。
- ・次の予習をします。

Appendix

im2colおよびcol2imに関する補足資料

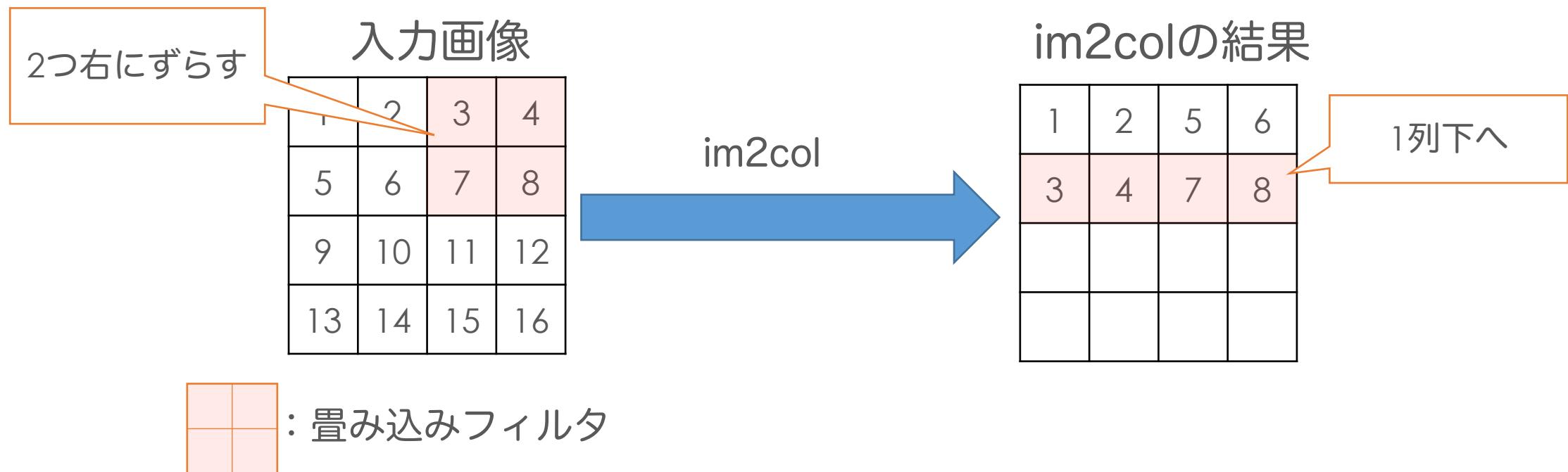
im2col

- image to columnsの略称で、畳み込み演算を一度の行列積で計算するために行う画像の変換方法のこと
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：



im2col

- Image to columnsの略称で、畳み込み演算を一度の行列積で計算するために行う画像の変換方法のこと
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：



im2col

- Image to columnsの略称で、畳み込み演算を一度の行列積で計算するために行う画像の変換方法のこと
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：

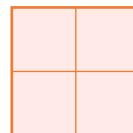
入力画像

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

im2col

im2colの結果

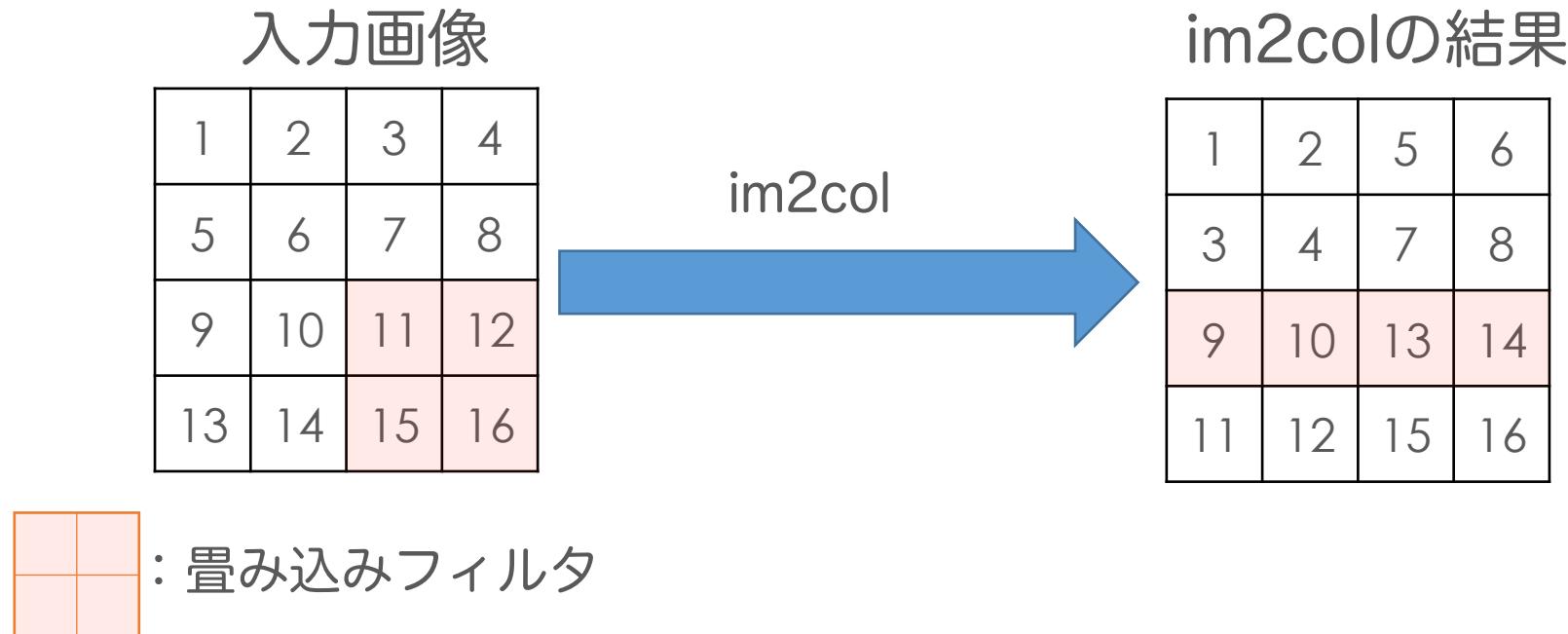
1	2	5	6
3	4	7	8
9	10	13	14



: 畳み込みフィルタ

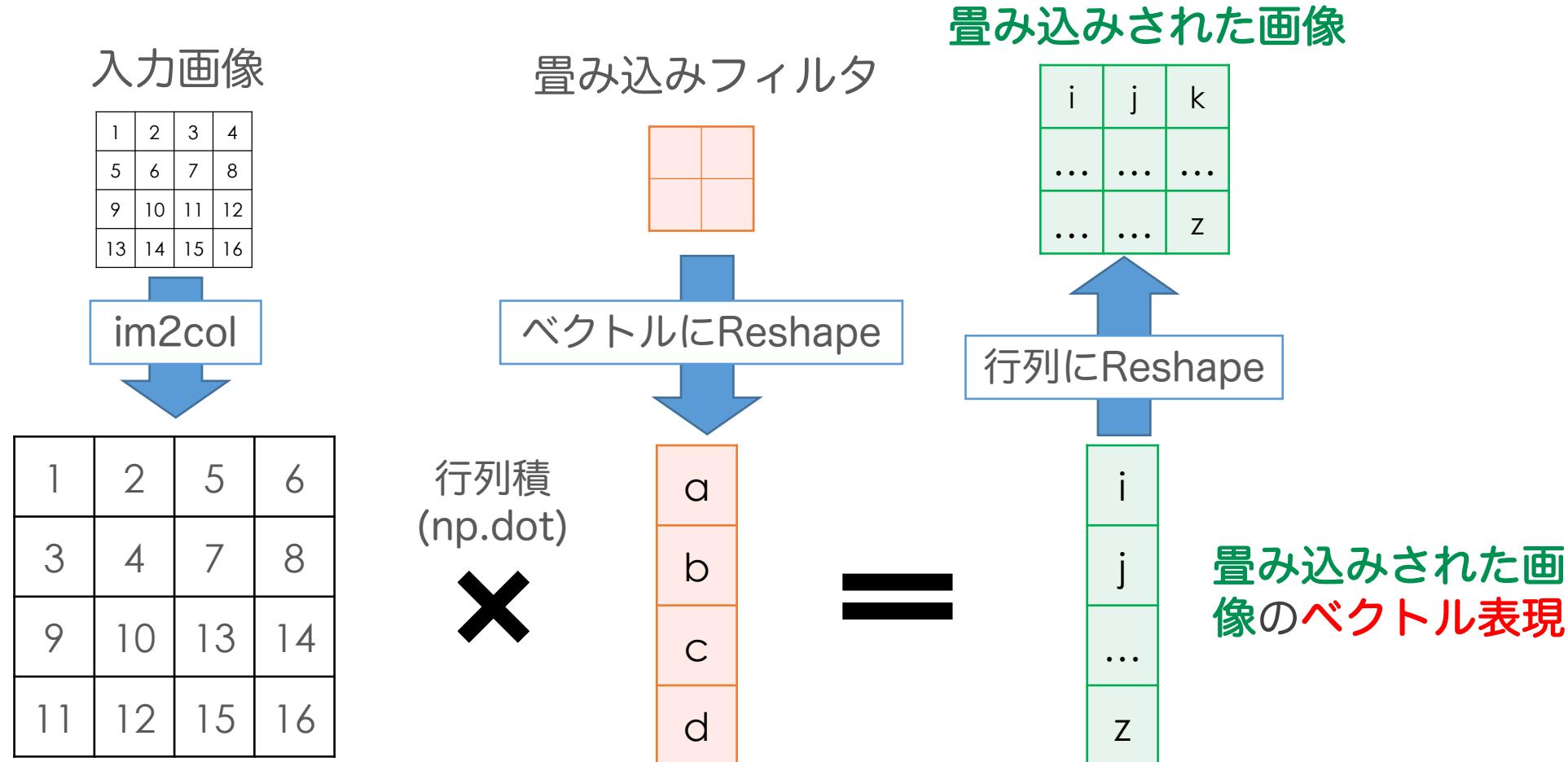
im2col

- Image to columnsの略称で、畳み込み演算を一度の行列積で計算するために行う画像の変換方法のこと
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：



im2colと畳み込み演算

- im2colによって得られた画像とベクトルに変換した畳み込みフィルタの積を取ると畳み込みの結果が得られる



im2colの多チャンネル拡張

- im2colを多チャンネルに拡張するときは、チャンネル数に応じて列数を増やせばよい

2チャンネルの入力画像
(3次元配列)

1	2	3	4
5	1	2	3
9	5	6	7
13	9	10	11
	13	14	15
		15	16

im2col

im2colの出力画像 (2次元配列)

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

im2colのバッチ処理への拡張

- さらにバッチ処理に拡張する場合は、バッチサイズに応じて行数を増やす

2チャンネルの入力画像2枚
(4次元配列)

1	2	3	4
5	1	2	3
9	5	6	7
13	9	10	11
13	14	15	16

1	2	3	4
5	1	2	3
9	5	6	7
13	9	10	11
13	14	15	16

im2col

im2colの出力画像 (2次元配列)

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16
1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

1枚目

2枚目

im2colの出力画像サイズ

- im2colの出力画像サイズは以下の式で求めることができる：

im2colの出力画像サイズ公式

出力画像の高さ： $H_{\text{im2col}} = B \times H_{\text{out}} \times W_{\text{out}}$

出力画像の幅： $W_{\text{im2col}} = C \times H_{\text{filter}} \times W_{\text{filter}}$

C は入力画像のチャンネル数、 $H_{\text{filter}}, W_{\text{filter}}$ はそれぞれフィルタの高さと幅、
 B はバッチサイズ、 $H_{\text{out}}, W_{\text{out}}$ はそれぞれ畳み込み後の出力画像の高さと幅を意味する

畳み込み・プーリング後の出力画像サイズ

- 畳み込みあるいはプーリング後の出力画像サイズの高さ H_{out} と幅 W_{out} は以下の式で求めることができる：

畳み込み後の出力画像サイズ公式

$$\text{出力画像の高さ} : H_{\text{out}} = \text{floor}\left(\frac{H+2p-H_{\text{filter}}}{s}\right) + 1$$

$$\text{出力画像の幅} : W_{\text{out}} = \text{floor}\left(\frac{W+2p-W_{\text{filter}}}{s}\right) + 1$$

H, W はそれぞれ入力画像の高さと幅、

$H_{\text{filter}}, W_{\text{filter}}$ はそれぞれフィルタの高さと幅、 p はパディングサイズ、 s はストライド、関数 floor は小数点以下を切り捨てる関数を意味する

スライスを用いたim2colの効率的な実装

- そのまま実装すると入力画像が大きくなると処理が増えて非効率
- Notebookではスライスを用いることで、im2colを効率的に実装している
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：

置き込みフィルタ
の左上の部分

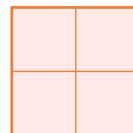
入力画像

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

im2col

im2colの出力

1			
3			
9			
11			



: 置き込みフィルタ

スライスを用いたim2colの効率的な実装

- そのまま実装すると入力画像が大きくなると処理が増えて非効率
- Notebookではスライスを用いることで、im2colを効率的に実装している
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：

置き込みフィルタ
の右上の部分

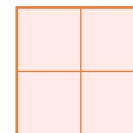
入力画像

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

im2col

im2colの出力

1	2		
3	4		
9	10		
11	12		



: 置き込みフィルタ

スライスを用いたim2colの効率的な実装

- そのまま実装すると入力画像が大きくなると処理が増えて非効率
- Notebookではスライスを用いることで、im2colを効率的に実装している
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：

置き込みフィルタ
の左下の部分

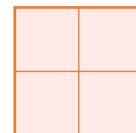
入力画像

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

im2col

im2colの出力

1	2	5	
3	4	7	
9	10	13	
11	12	15	



: 置き込みフィルタ

スライスを用いたim2colの効率的な実装

- そのまま実装すると入力画像が大きくなると処理が増えて非効率
- Notebookではスライスを用いることで、im2colを効率的に実装している
- 画像サイズ4x4、フィルタサイズ2x2、ストライド2での例：

畳み込みフィルタ
の右下の部分

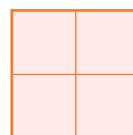
入力画像

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

im2col

im2colの出力

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

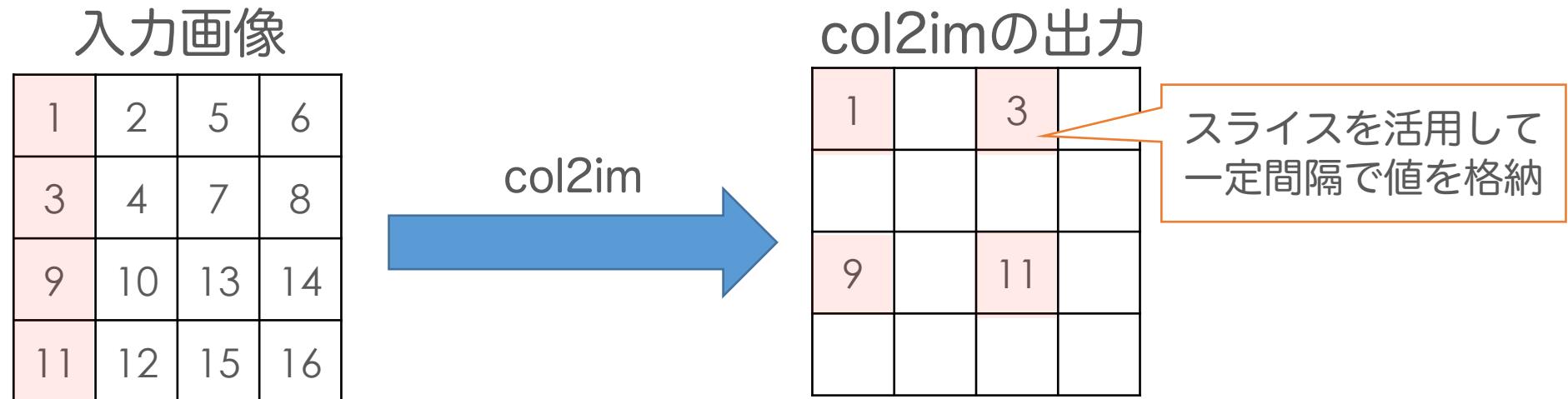


: 畳み込みフィルタ

スライスを用いれば画像のサイズに依らずに
フィルタの要素数回の処理でim2colが実装可能

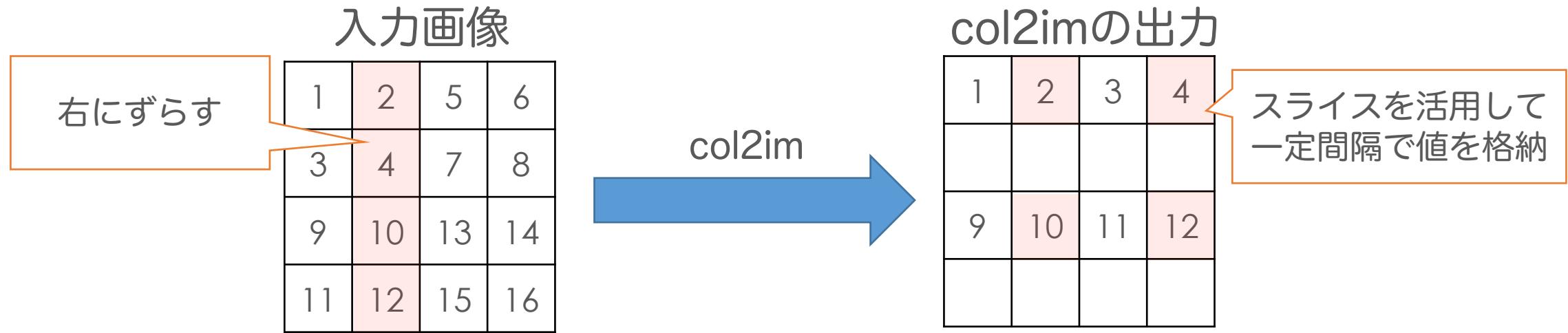
col2im

- Columns to imageの略称で、im2colの逆操作として定義
- im2colが順伝播計算、col2imが逆伝播計算のときにそれぞれ使用される
- im2colと同じくスライスによって効率的に実装可能



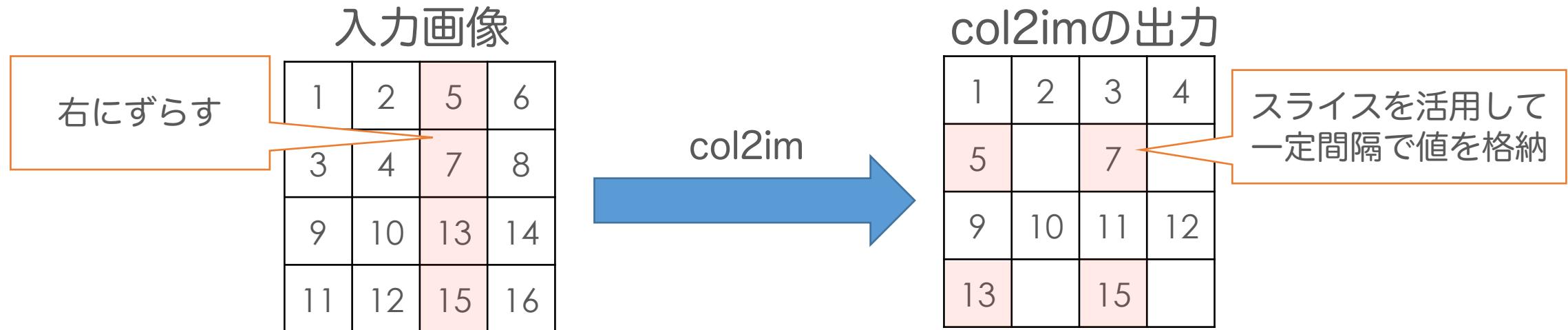
col2im

- Columns to imageの略称で、im2colの逆操作として定義
- Im2colが順伝播、col2imが逆伝播のときにそれぞれ使用される
- Im2colと同じくスライスによって効率的に実装可能



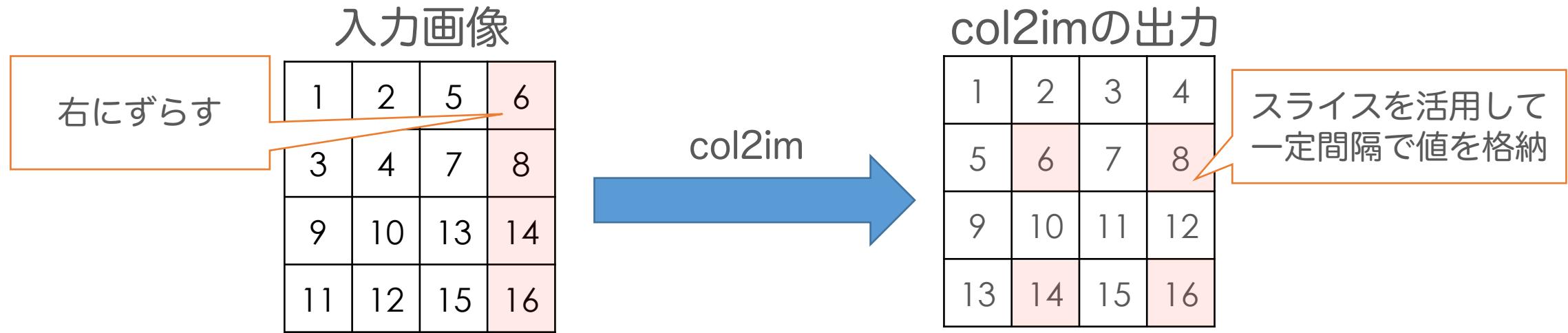
col2im

- Columns to imageの略称で、Im2colの逆操作として定義
- Im2colが順伝播、Col2imが逆伝播のときにそれぞれ使用される
- Im2colと同じくスライスによって効率的に実装可能



col2im

- Columns to imageの略称で、Im2colの逆操作として定義
- Im2colが順伝播、Col2imが逆伝播のときにそれぞれ使用される
- Im2colと同じくスライスによって効率的に実装可能

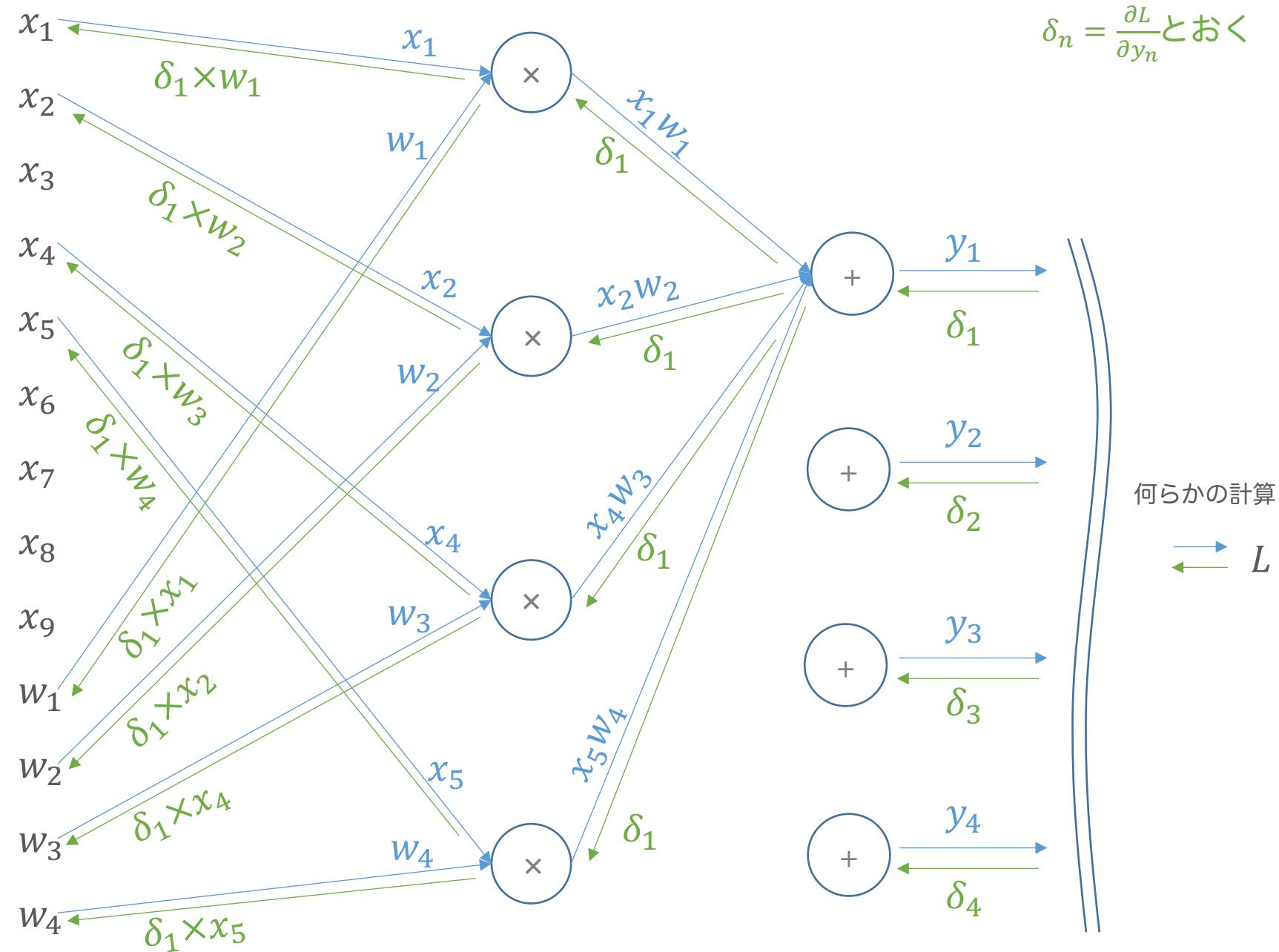


畳み込みレイヤおよびマックスプーリングレイヤ に関する補足資料

畳み込みレイヤ

- 畳み込みレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤが参考になる。

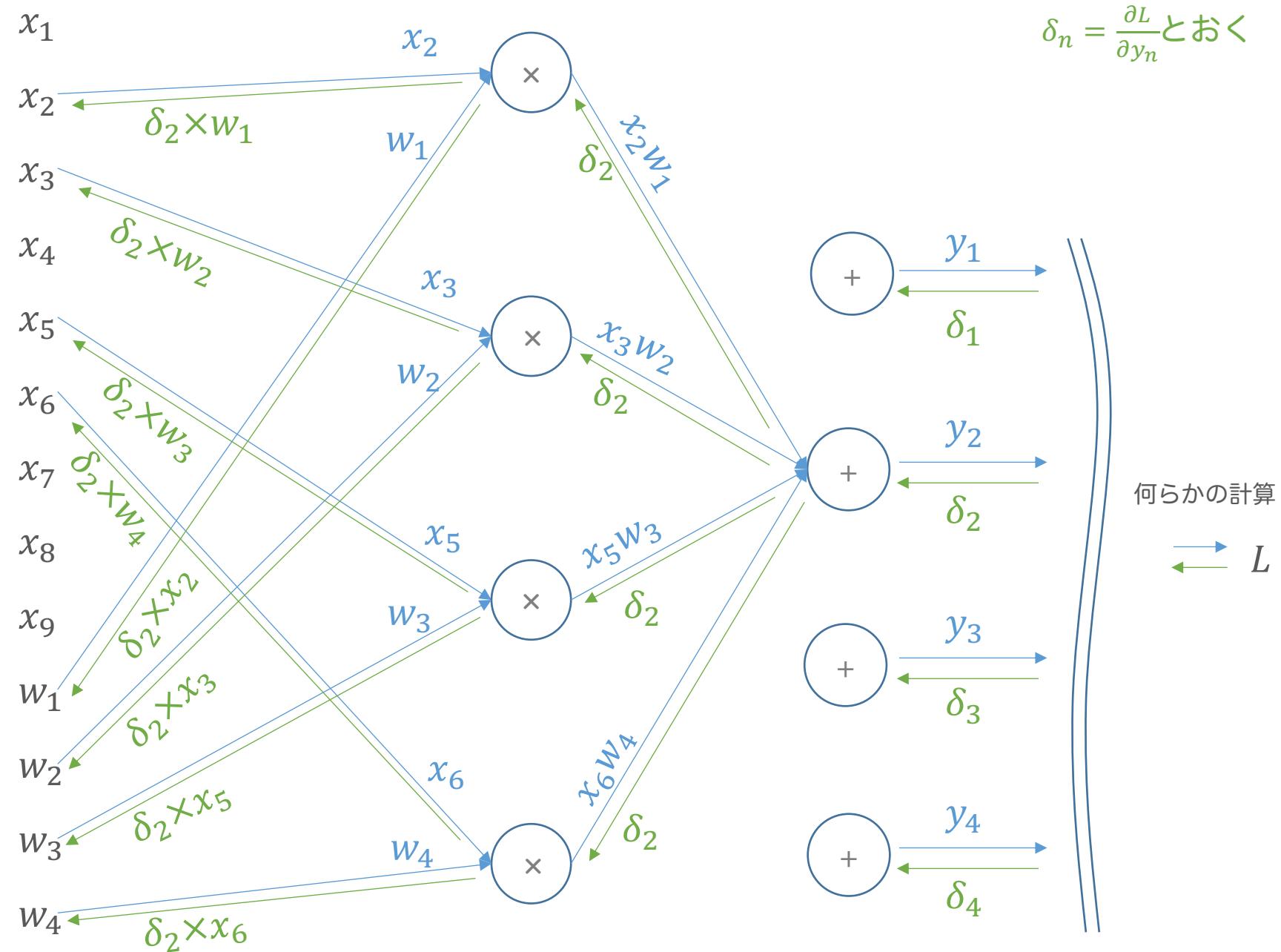
<条件>
入力データの高さ=3
入力データの幅=3
フィルタの高さ=2
フィルタの幅=2
ストライド=1
パッド=0
バイアスは省略



畳み込みレイヤ

- 畳み込みレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤが参考になる。

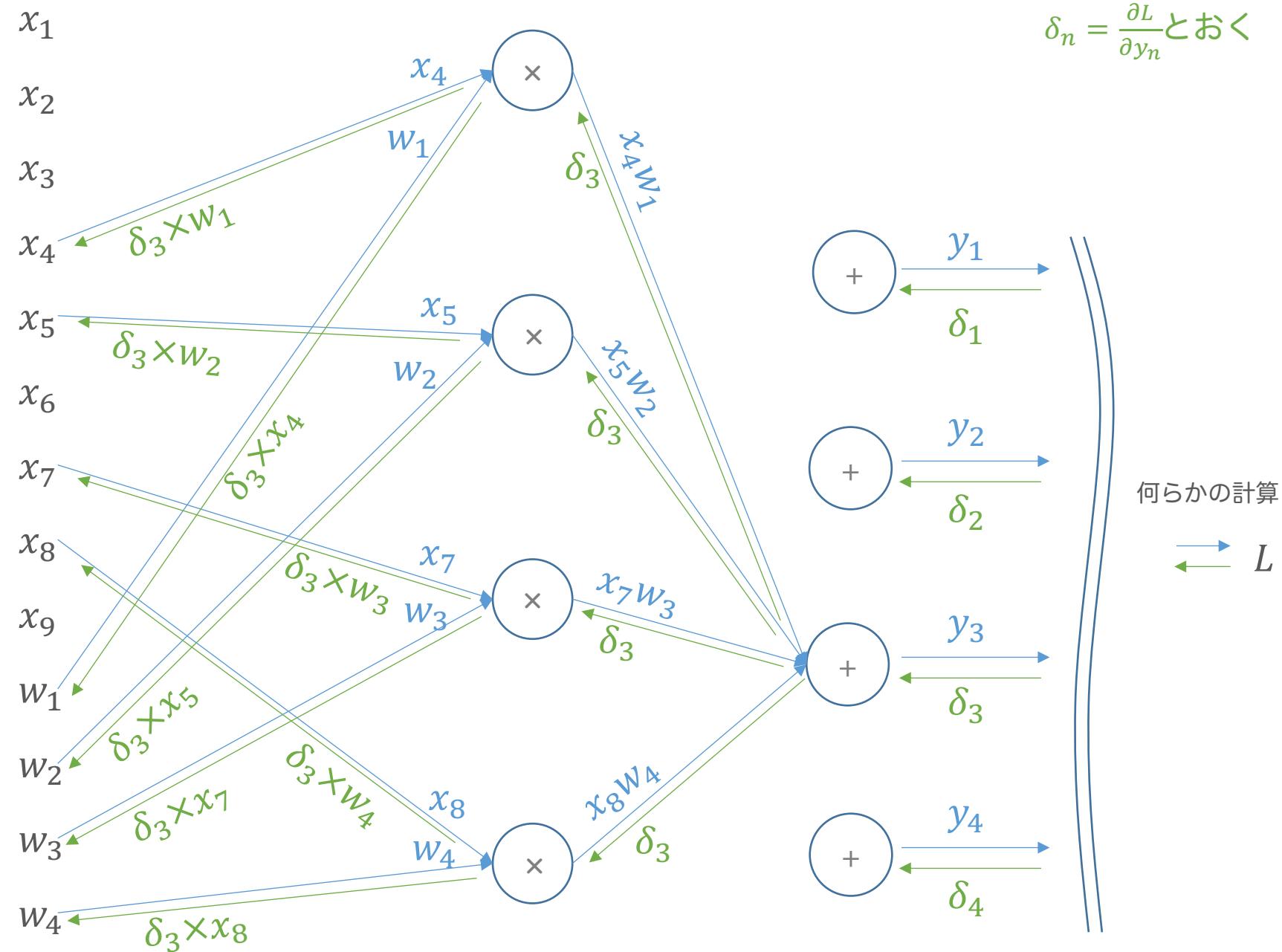
<条件>
入力データの高さ=3
入力データの幅=3
フィルタの高さ=2
フィルタの幅=2
ストライド=1
パッド=0
バイアスは省略



畳み込みレイヤ

- 畳み込みレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤが参考になる。

<条件>
 入力データの高さ=3
 入力データの幅=3
 フィルタの高さ=2
 フィルタの幅=2
 ストライド=1
 パッド=0
 バイアスは省略



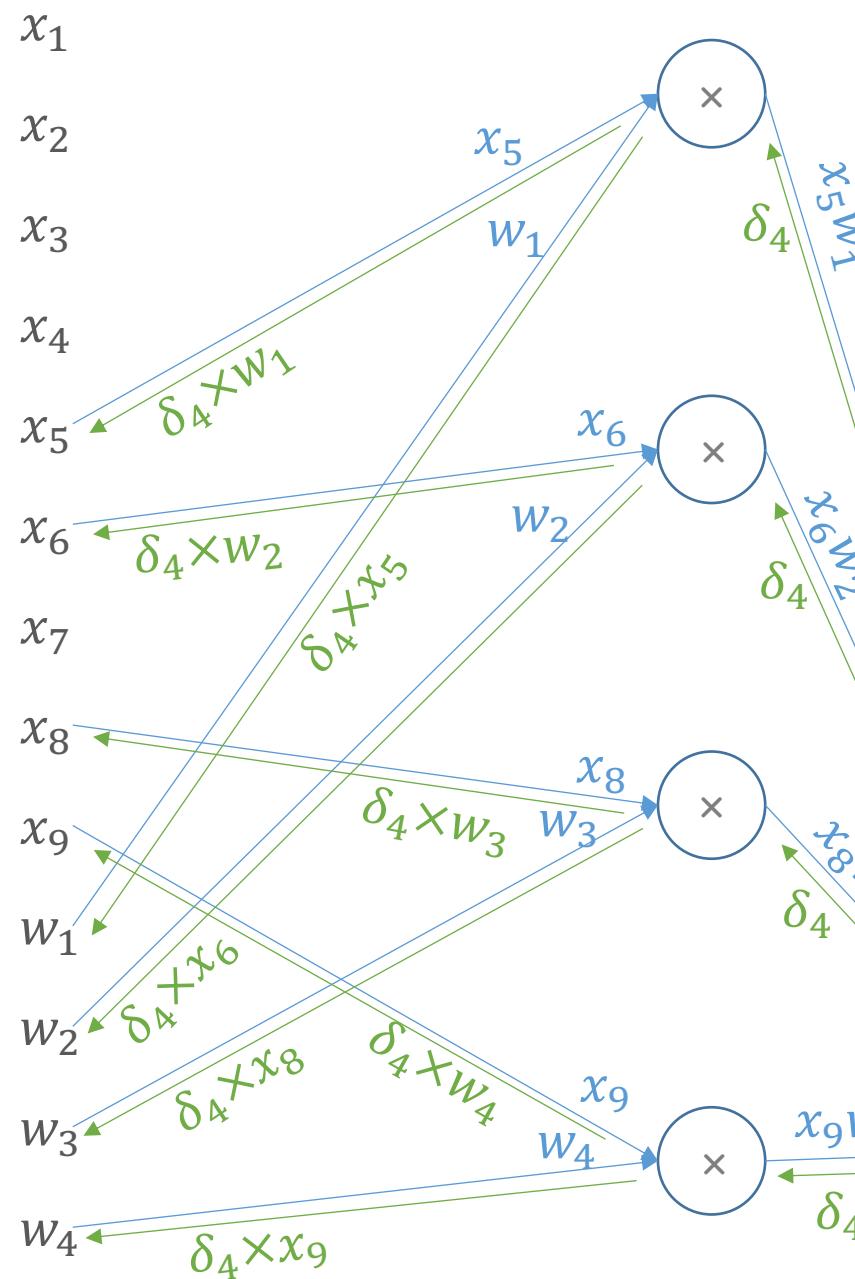
$$\delta_n = \frac{\partial L}{\partial y_n}$$

何らかの計算
 $\longleftrightarrow L$

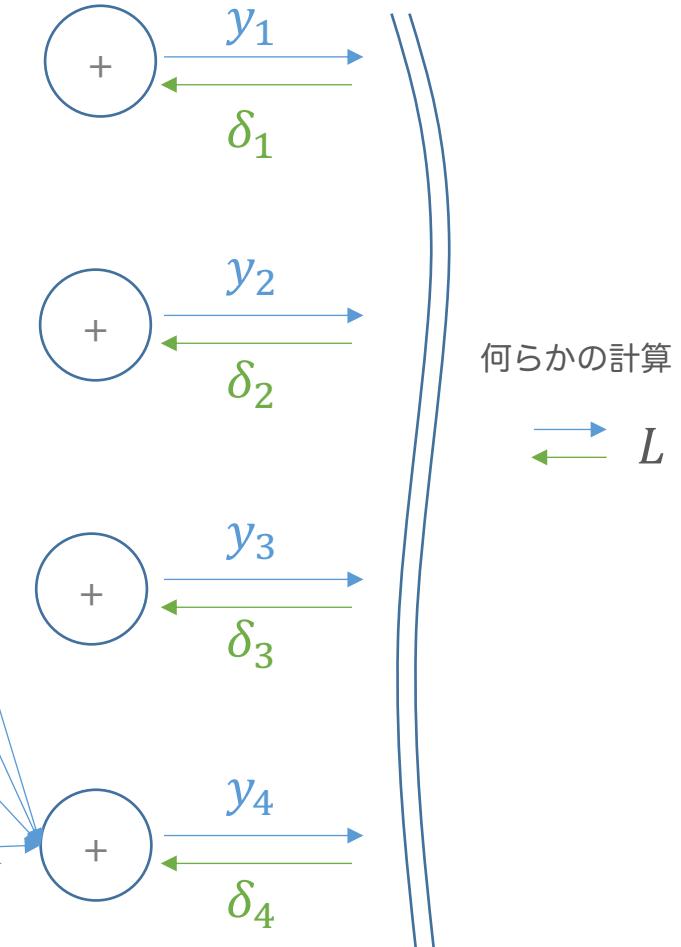
畳み込みレイヤ

- 畳み込みレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤが参考になる。

<条件>
入力データの高さ=3
入力データの幅=3
フィルタの高さ=2
フィルタの幅=2
ストライド=1
パット=0
バイアスは省略



$$\delta_n = \frac{\partial L}{\partial y_n}$$
とおく



何らかの計算

L

畳み込みレイヤ

- 畳み込みレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤが参考になる。

<条件>

入力データの高さ=3
入力データの幅=3
フィルタの高さ=2
フィルタの幅=2
ストライド=1
パット=0
バイアスは省略

$$\begin{aligned}x_1 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_1 \times w_1} \\x_2 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_1 \times w_2 + \delta_2 \times w_1} \\x_3 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_2 \times w_2} \\x_4 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_1 \times w_3 + \delta_3 \times w_1} \\x_5 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_1 \times w_4 + \delta_2 \times w_3 + \delta_3 \times w_2 + \delta_4 \times w_1} \\x_6 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_2 \times w_4 + \delta_4 \times w_2} \\x_7 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_3 \times w_3} \\x_8 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_3 \times w_4 + \delta_4 \times w_3} \\x_9 &\xrightarrow{\frac{\partial L}{\partial x_1} = \delta_4 \times w_4} \\w_1 &\xrightarrow{\frac{\partial L}{\partial w_1} = \delta_1 \times x_1 + \delta_2 \times x_2 + \delta_3 \times x_4 + \delta_4 \times x_5} \\w_2 &\xrightarrow{\frac{\partial L}{\partial w_2} = \delta_1 \times x_2 + \delta_2 \times x_3 + \delta_3 \times x_5 + \delta_4 \times x_6} \\w_3 &\xrightarrow{\frac{\partial L}{\partial w_3} = \delta_1 \times x_4 + \delta_2 \times x_5 + \delta_3 \times x_7 + \delta_4 \times x_8} \\w_4 &\xrightarrow{\frac{\partial L}{\partial w_4} = \delta_1 \times x_5 + \delta_2 \times x_6 + \delta_3 \times x_8 + \delta_4 \times x_9}\end{aligned}$$

x_n 毎、 w_n 毎に勾配を足し合わせる

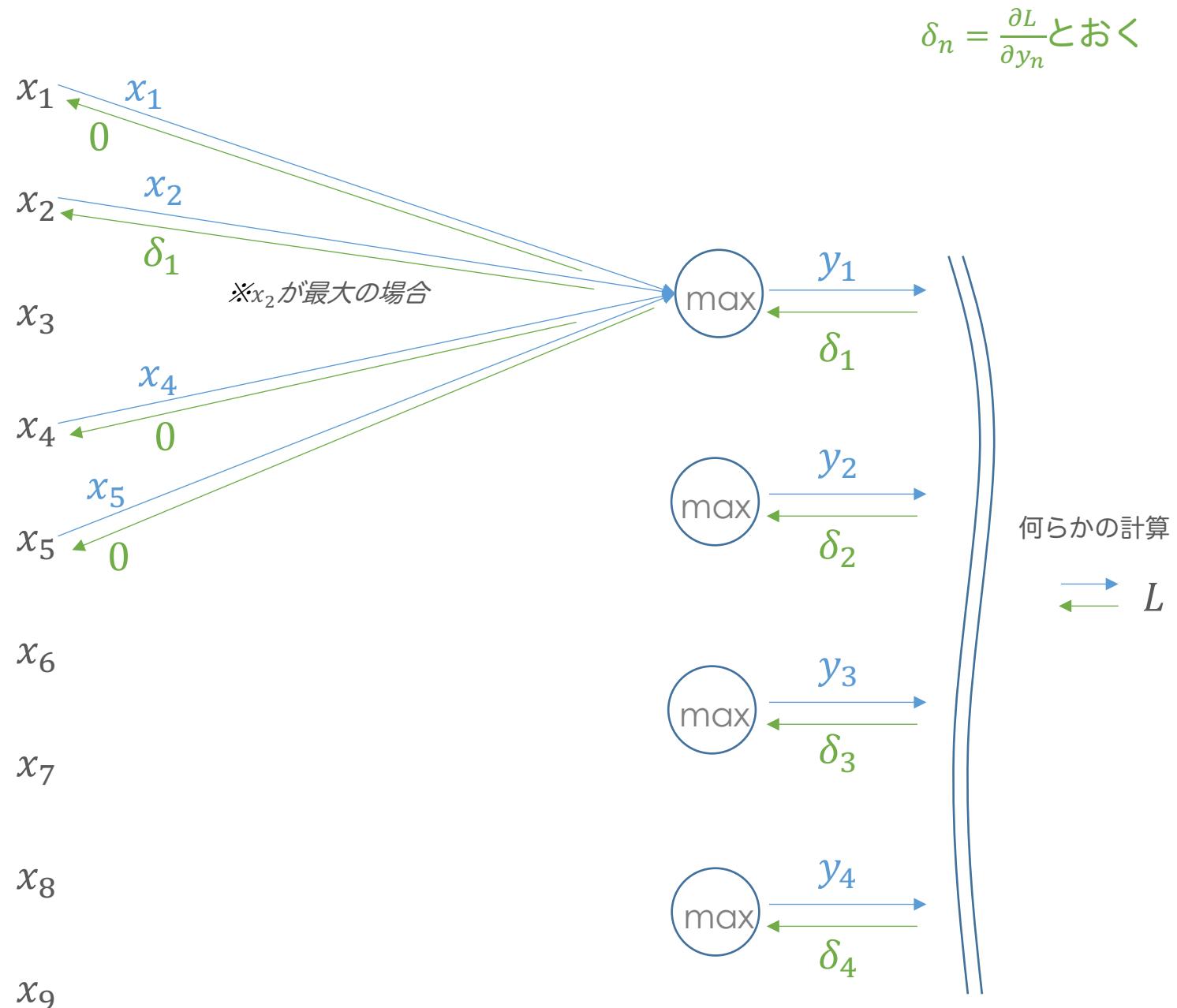
マックスプーリングレイヤ

$$\delta_n = \frac{\partial L}{\partial y_n}$$
とおく

- マックスプーリングレイヤの計算グラフ例を右記に示す。
- バッチ版の考え方は、アフィンレイヤとドロップアウトレイヤが参考になる。

<条件>

入力データの高さ=3
入力データの幅=3
プーリング窓の高さ=2
プーリング窓の幅=2
ストライド=1
パッド=0



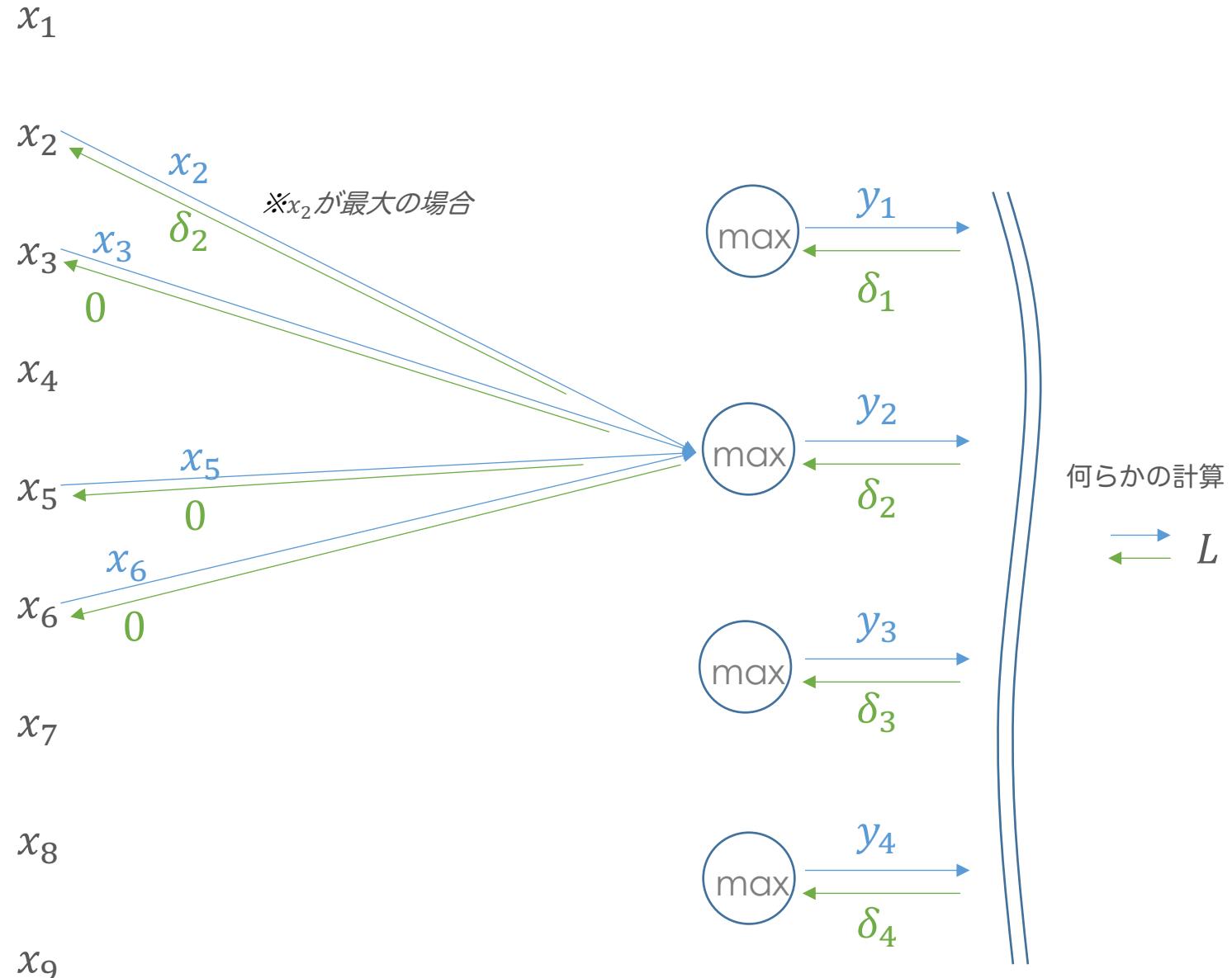
マックスプーリングレイヤ

$$\delta_n = \frac{\partial L}{\partial y_n}$$
とおく

- マックスプーリング
レイヤの計算グラフ
例を右記に示す。
- バッチ版の考え方は、
アフィンレイヤとド
ロップアウトレイヤ
が参考になる。

<条件>

入力データの高さ=3
入力データの幅=3
プーリング窓の高さ=2
プーリング窓の幅=2
ストライド=1
パッド=0

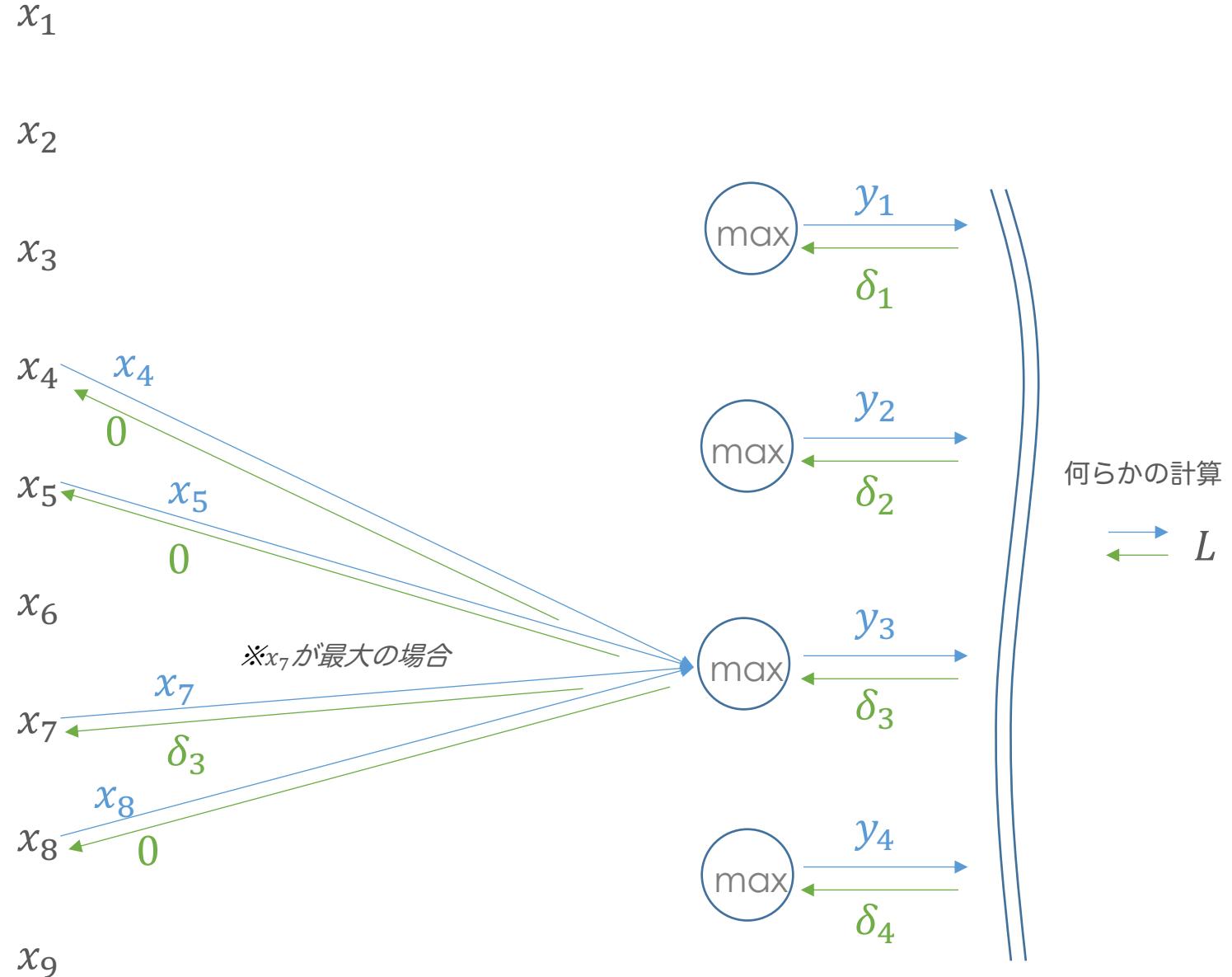


マックスプーリングレイヤ

$$\delta_n = \frac{\partial L}{\partial y_n}$$
とおく

- マックスプーリング
レイヤの計算グラフ
例を右記に示す。
- バッチ版の考え方は、
アフィンレイヤとド
ロップアウトレイヤ
が参考になる。

<条件>
入力データの高さ=3
入力データの幅=3
プーリング窓の高さ=2
プーリング窓の幅=2
ストライド=1
パッド=0

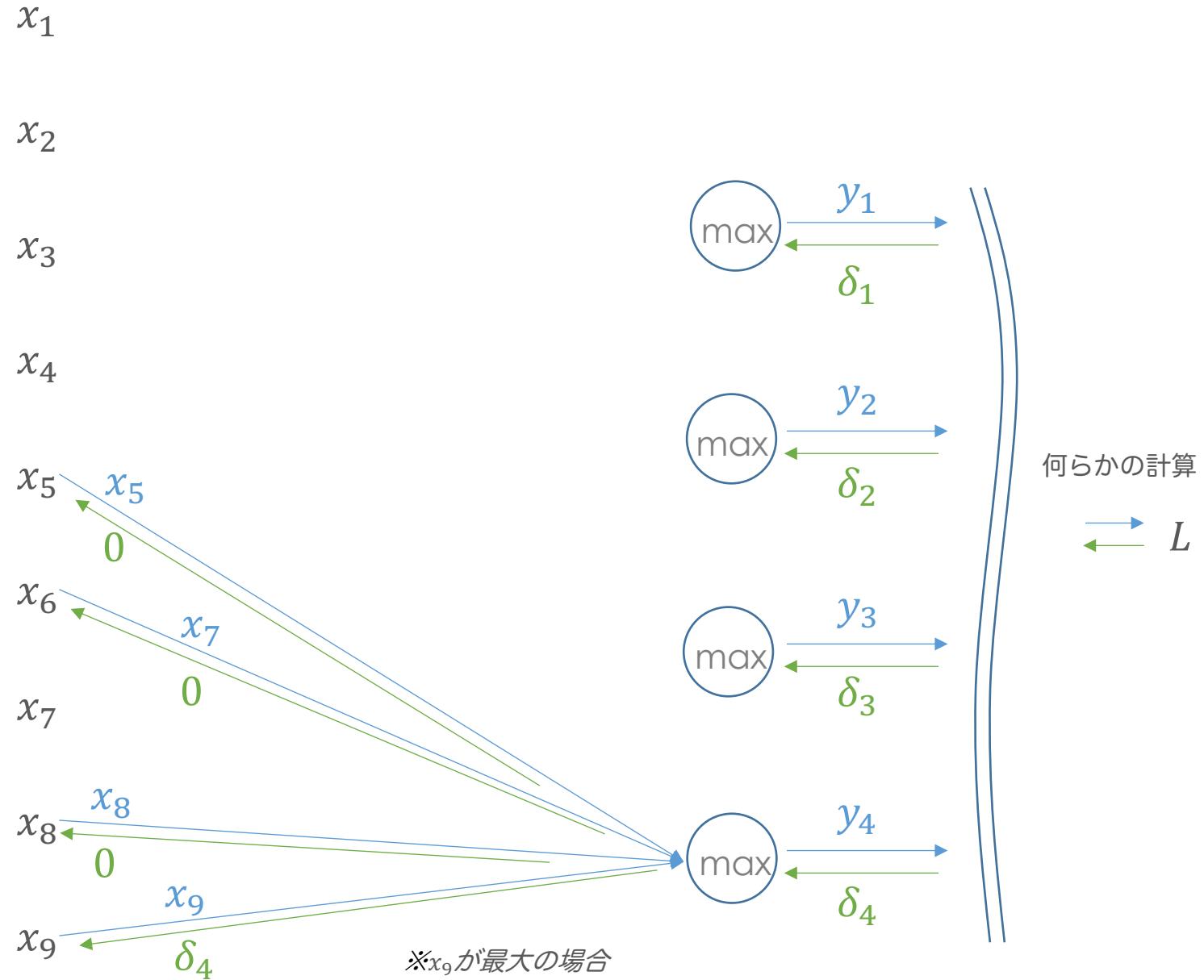


マックスプーリングレイヤ

$$\delta_n = \frac{\partial L}{\partial y_n}$$
とおく

- マックスプーリング
レイヤの計算グラフ
例を右記に示す。
- バッチ版の考え方は、
アフィンレイヤとド
ロップアウトレイヤ
が参考になる。

<条件>
入力データの高さ=3
入力データの幅=3
プーリング窓の高さ=2
プーリング窓の幅=2
ストライド=1
パッド=0



マックスプーリングレイヤ

- マックスプーリング
レイヤの計算グラフ
例を右記に示す。
- バッチ版の考え方は、
アフィンレイヤとド
ロップアウトレイヤ
が参考になる。

<条件>

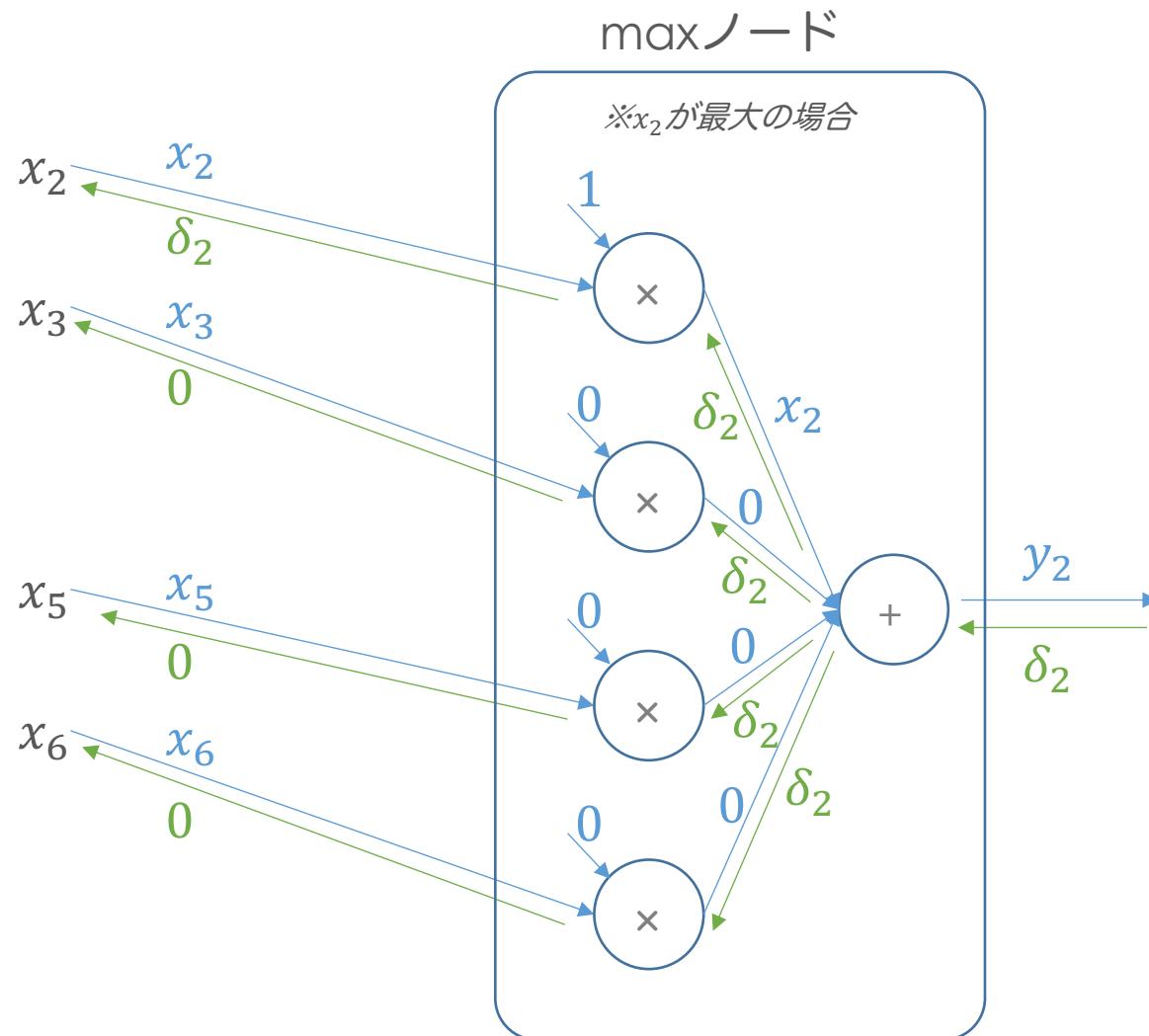
入力データの高さ=3
入力データの幅=3
プーリング窓の高さ=2
プーリング窓の幅=2
ストライド=1
パッド=0

$$x_1 \xrightarrow[\partial L / \partial x_1 = 0]{x_1}$$
$$x_2 \xrightarrow[\partial L / \partial x_2 = \delta_1 + \delta_2]{x_2}$$
$$x_3 \xrightarrow[\partial L / \partial x_3 = 0]{x_3}$$
$$x_4 \xrightarrow[\partial L / \partial x_4 = 0 + 0]{x_4}$$
$$x_5 \xrightarrow[\partial L / \partial x_5 = 0 + 0 + 0 + 0]{x_5}$$
$$x_6 \xrightarrow[\partial L / \partial x_6 = 0 + 0]{x_6}$$
$$x_7 \xrightarrow[\partial L / \partial x_7 = \delta_3]{x_7}$$
$$x_8 \xrightarrow[\partial L / \partial x_8 = 0 + 0]{x_8}$$
$$x_9 \xrightarrow[\partial L / \partial x_9 = \delta_4]{x_9}$$

x_n 毎に勾配を足し合わせる

マックスプーリングレイヤ

- maxノードでの計算を
詳細に書くと右記の
ようになる。



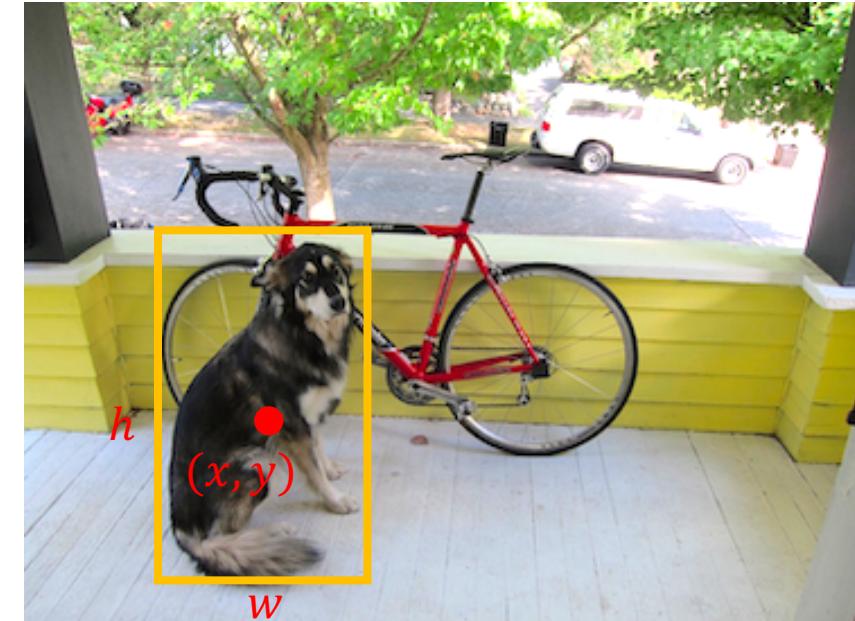
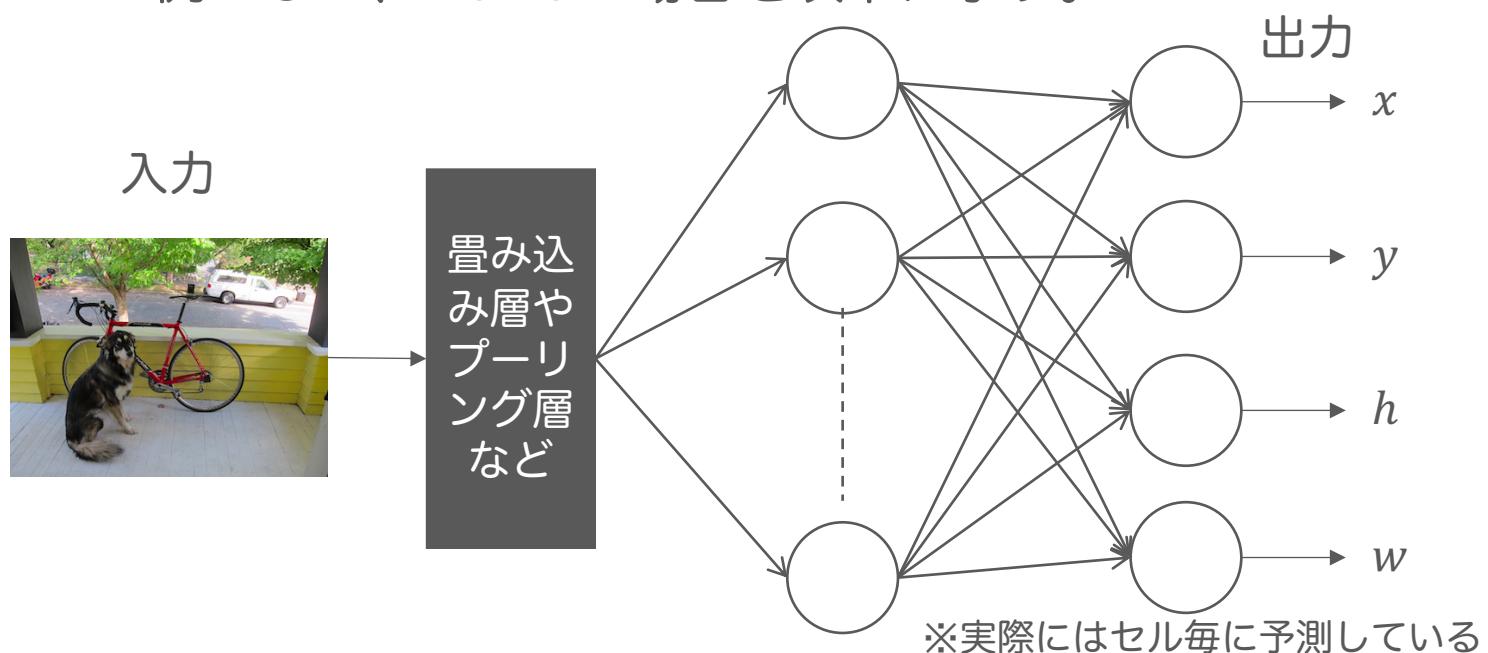
Any Questions?

Appendix

物体検出モデルにおけるバウンディングボックスの回帰

物体検出モデルにおけるバウンディングボックスの回帰

- 物体検出モデルでよく出てくる「バウンディングボックスを回帰する」とは、NNによって求められた特徴量から、バウンディングボックスの位置と大きさを定めるための4つの数値を予測するという意味である。
- 例として、YOLOの場合を以下に示す。



x : バウンディングボックスの中心の x 座標^{*1}
 y : バウンディングボックスの中心の y 座標^{*1}
 h : バウンディングボックスの高さ
 w : バウンディングボックスの幅

^{*1} 学習データにおいては画像全体の相対座標 (0~1の値)。予測する際は、セル毎の相対座標 (0~1の値)を求めてから、画像全体の相対座標 (0~1の値)に変換している。