自然言語処理
—CNN —

https://satoyoshiharu.github.io/nlp/

100本ノック第9章とCNNの位置づけ

- １００本ノック課題集第9章は、RNN、CNN、Transformerを扱っている。
- CNNは、現在、画像処理系では基本となる技術である。100本ノックの課題では、自然言語処理へ応用する事例が載せられている。

自然言語処理
CNN

解説動画

Convolution層　(CNN)

特徴を抽出する(2次元データの場合)

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

入力

＊

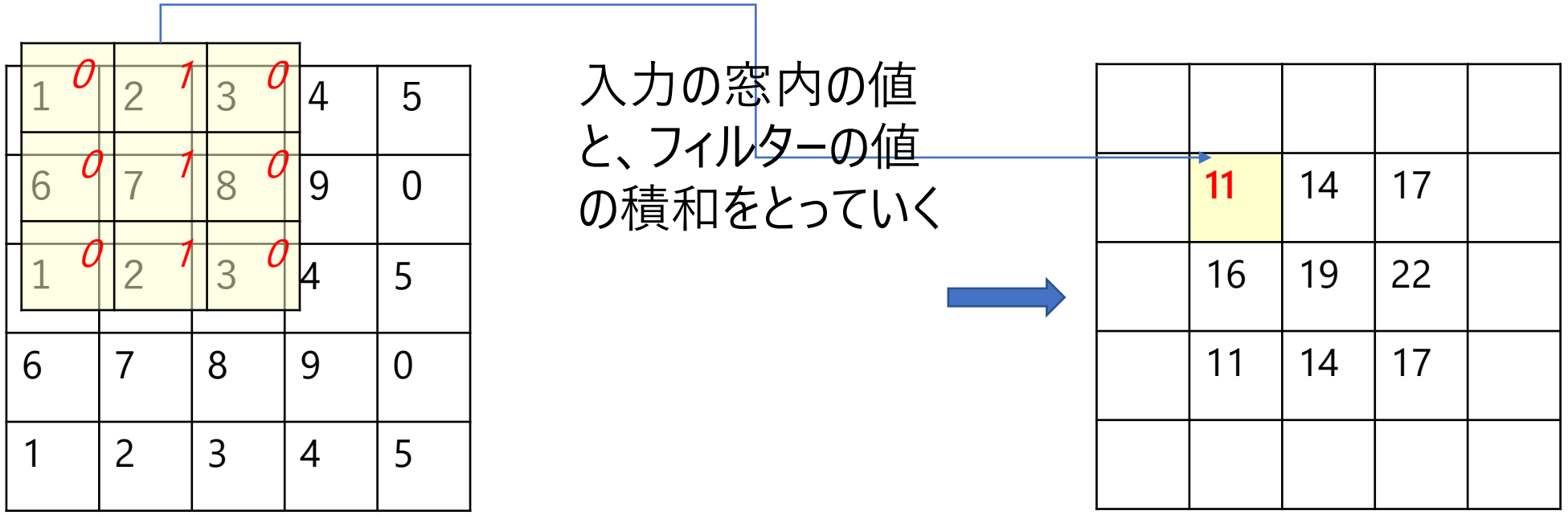
0	*1*	*0*
0	*1*	*0*
0	*1*	*0*

フィルター

	11	14	17	
	16	19	22	
	11	14	17	

出力

Convolution層



Convolution層

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

入力の窓内の値
と、フィルターの値
の積和をとっていく

	11	**14**	17	
	16	19	22	
	11	14	17	

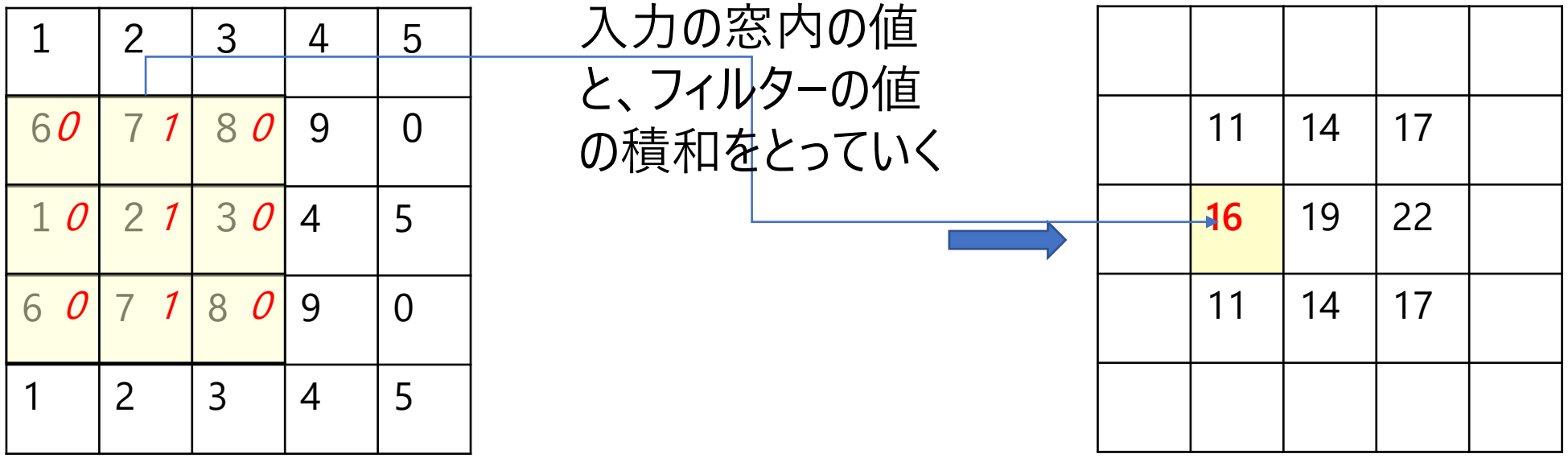
Convolution層

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

入力の窓内の値と、フィルターの値の積和をとっていく

	11	14	**17**	
	16	19	22	
	11	14	17	

Convolution層



以下省略...

フィルタの各要素が学習パラメータ

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

入力

*

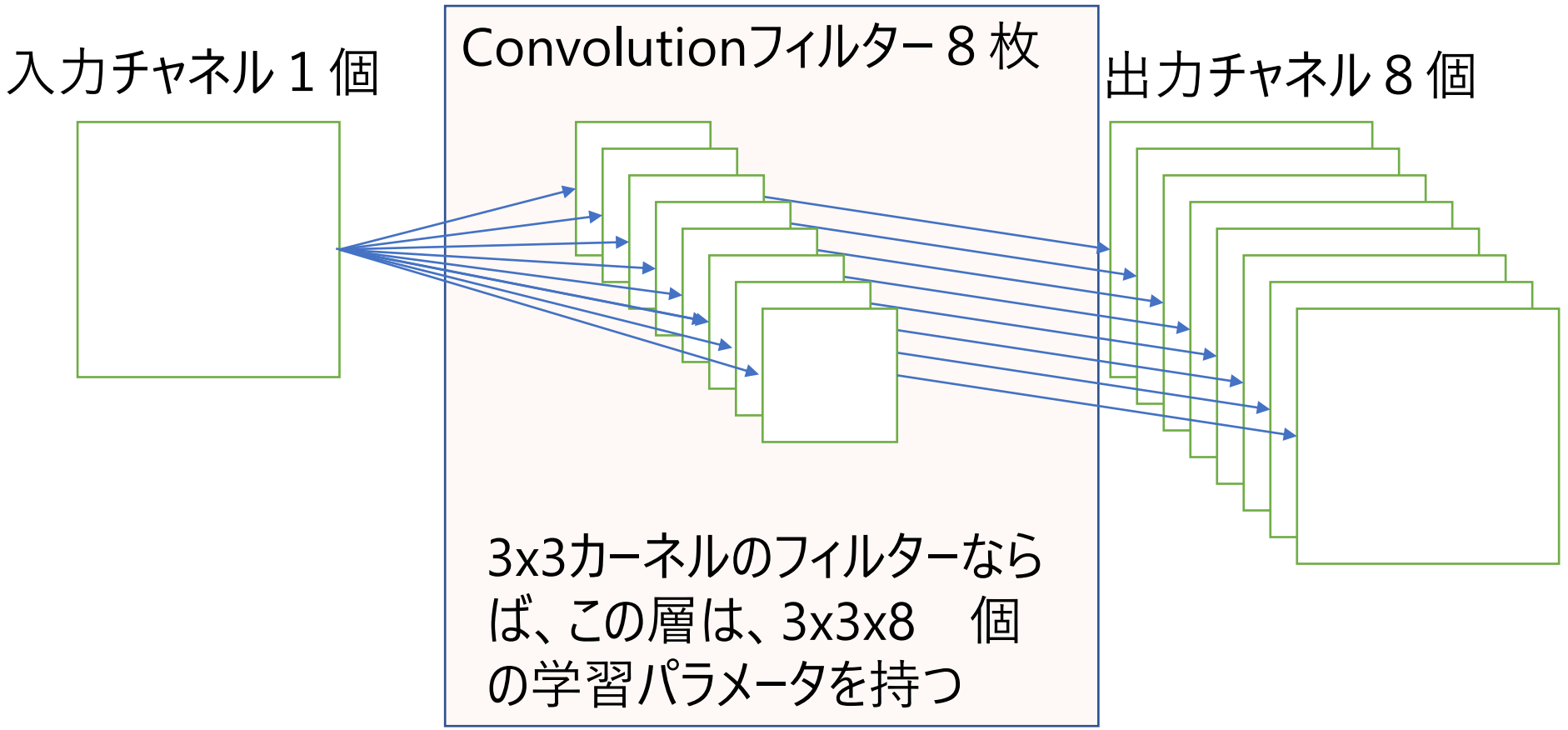
k11	*k12*	*K12*
k21	*k22*	*K23*
k31	*k32*	*k33*

フィルター

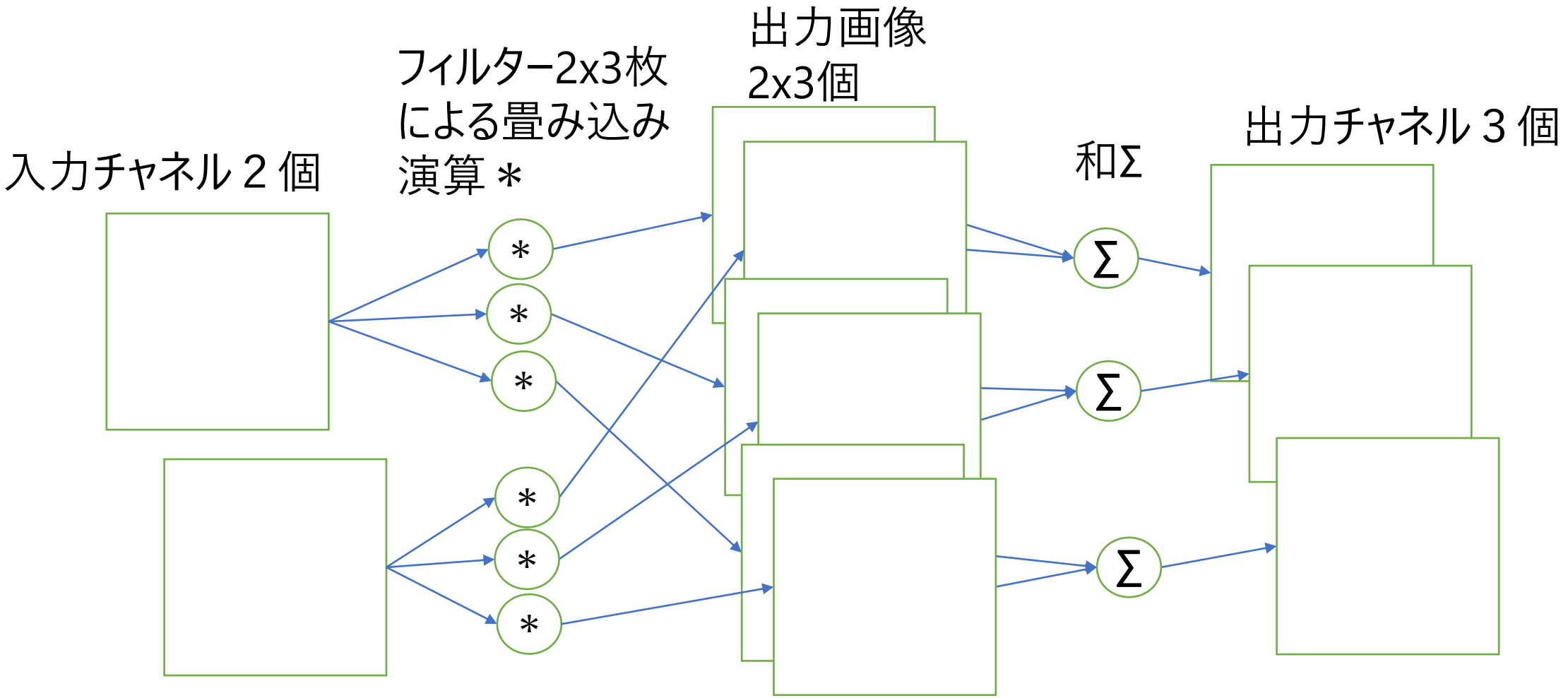
	11	14	17	
	16	19	22	
	11	14	17	

出力

Convolution層の入出力チャネルとパラメータ

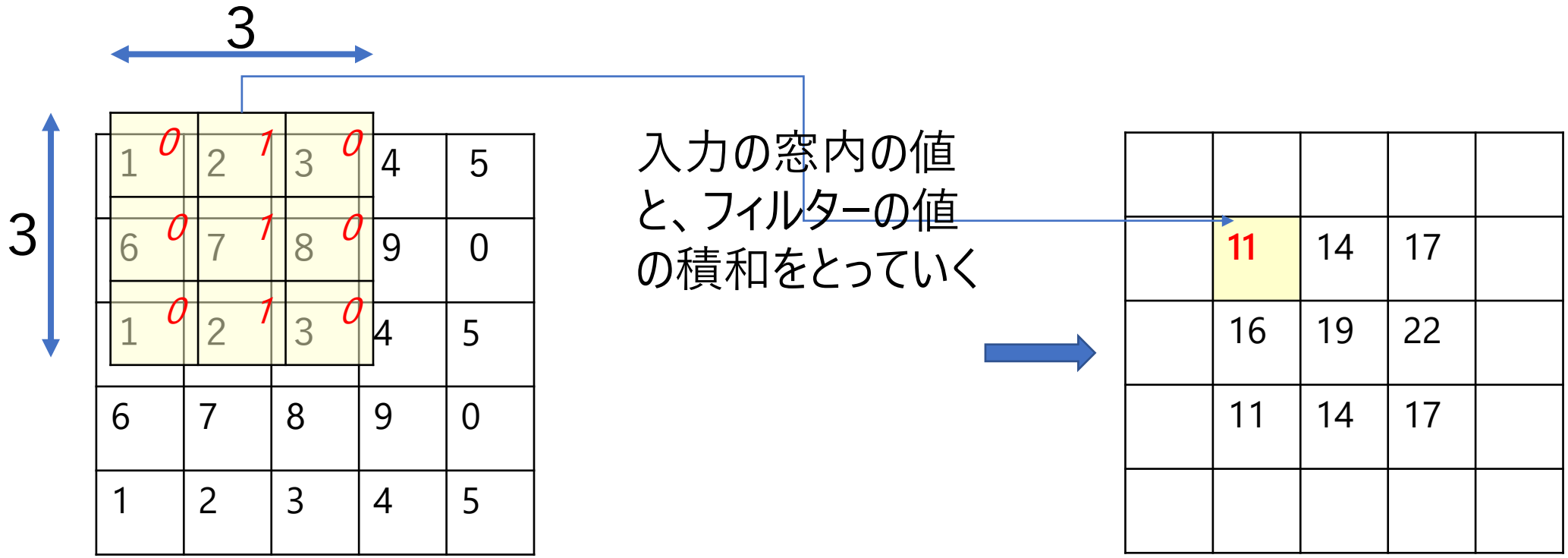


入力チャネルが複数の場合



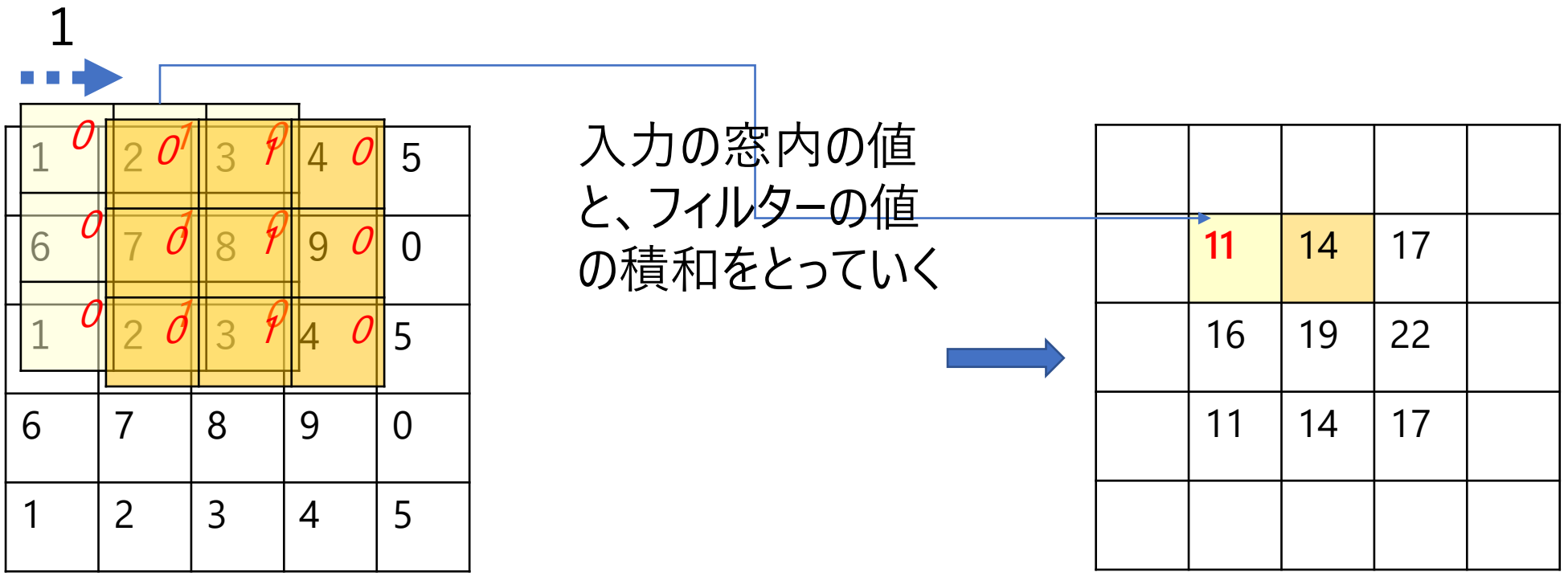
カーネルサイズ

3x3のカーネル



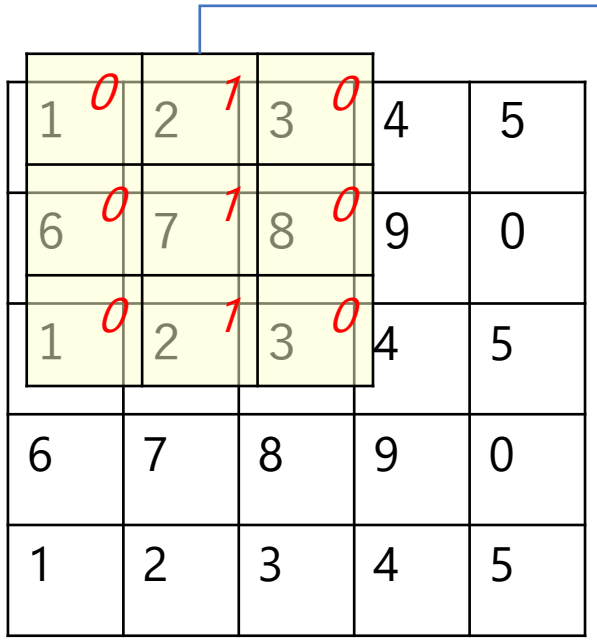
カーネルサイズが大きいほど、グローバルに観察できるが、学習パラメータが増える

ストライド（歩幅、移動幅）



ストライドが小さいほどきめ細かに観察するが、計算量が増える。

パディング



入力の窓内の値
と、フィルターの値
の積和をとっていく

11	14	17
16	19	22
11	14	17

Convolutionはデフォルトで出力は小さくなる。

パディング

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

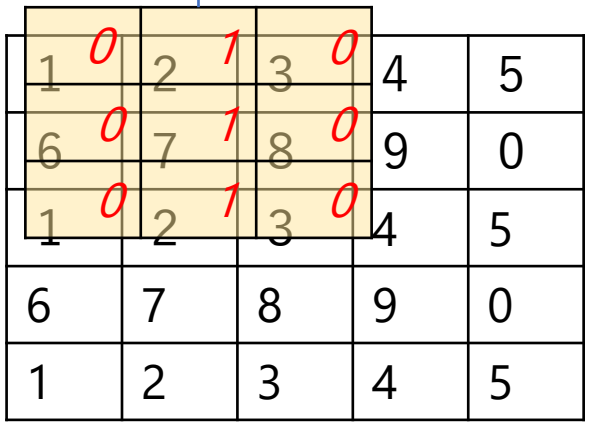
入力の窓内の値
と、フィルターの値
の積和をとっていく

7	9	11	13	5
8	**11**	14	17	10
13	16	19	22	5
8	11	14	17	10
7	9	11	13	5

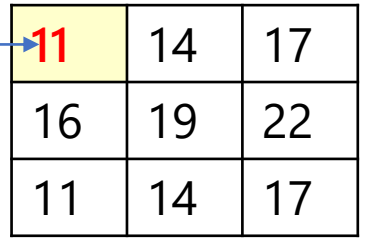
外境界上のConvolutionに関し、入力の外側に0とかを想定して計算し、出力を同じサイズにする。

ディレーション(拡張、伸長)

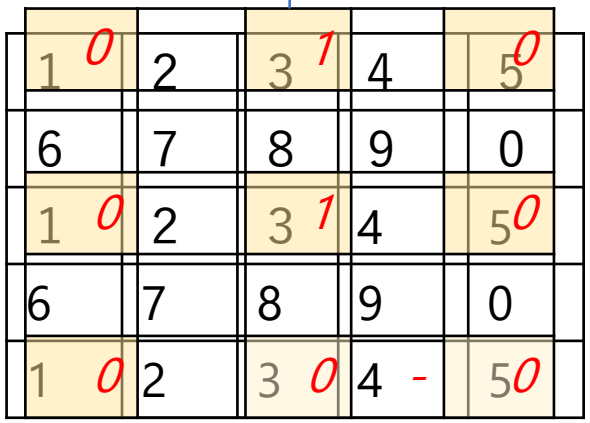
ディレーション=1



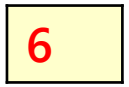
入力の窓内の値と、フィルターの値の積和をとっていく



ディレーション=2



入力の窓内の値と、フィルターの値の積和をとっていく



よりグローバルに入力を観察して、高速に出力を得る

CNNの貢献と今

- CNNはDeep Learningの発展を支えた。
- 今、Self-AttentionがCNNに影響を及ぼしている。

自然言語処理
MaxPooling

解説動画

MaxPooling

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

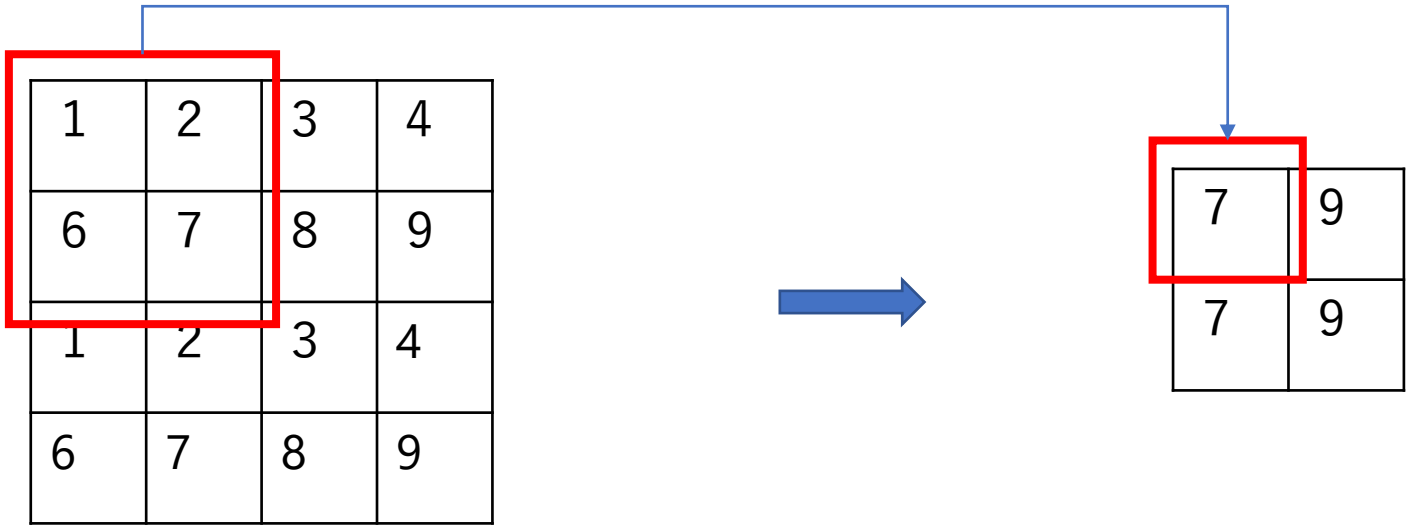
7	9
7	9

入力

出力

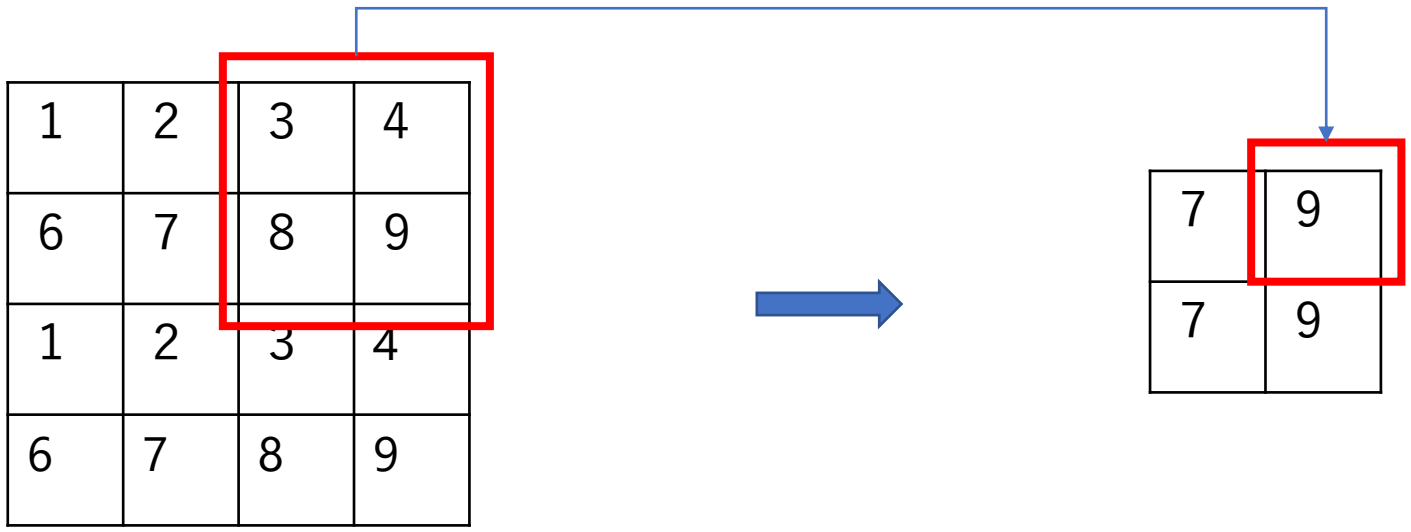
MaxPooling

最も大きな値を拾っていく



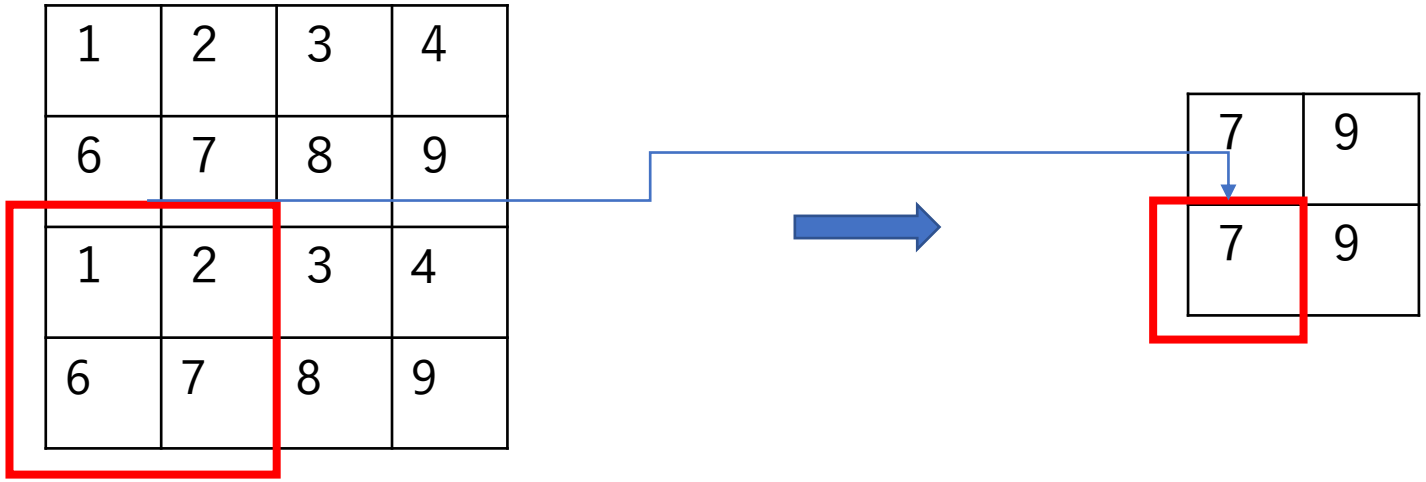
MaxPooling

最も大きな値を拾っていく



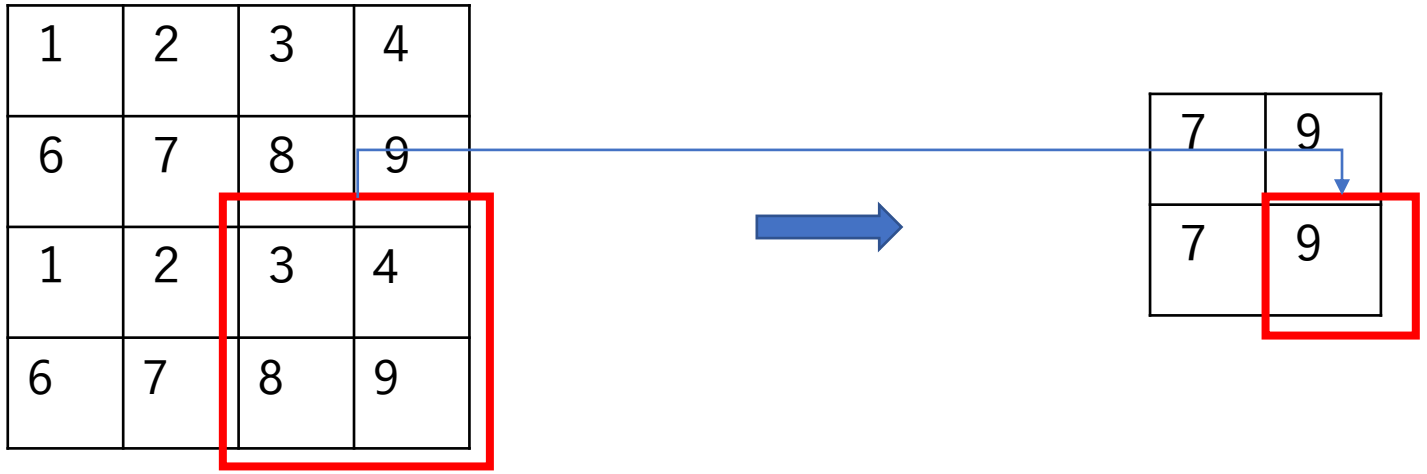
MaxPooling

最も大きな値を拾っていく



MaxPooling

最も大きな値を拾っていく



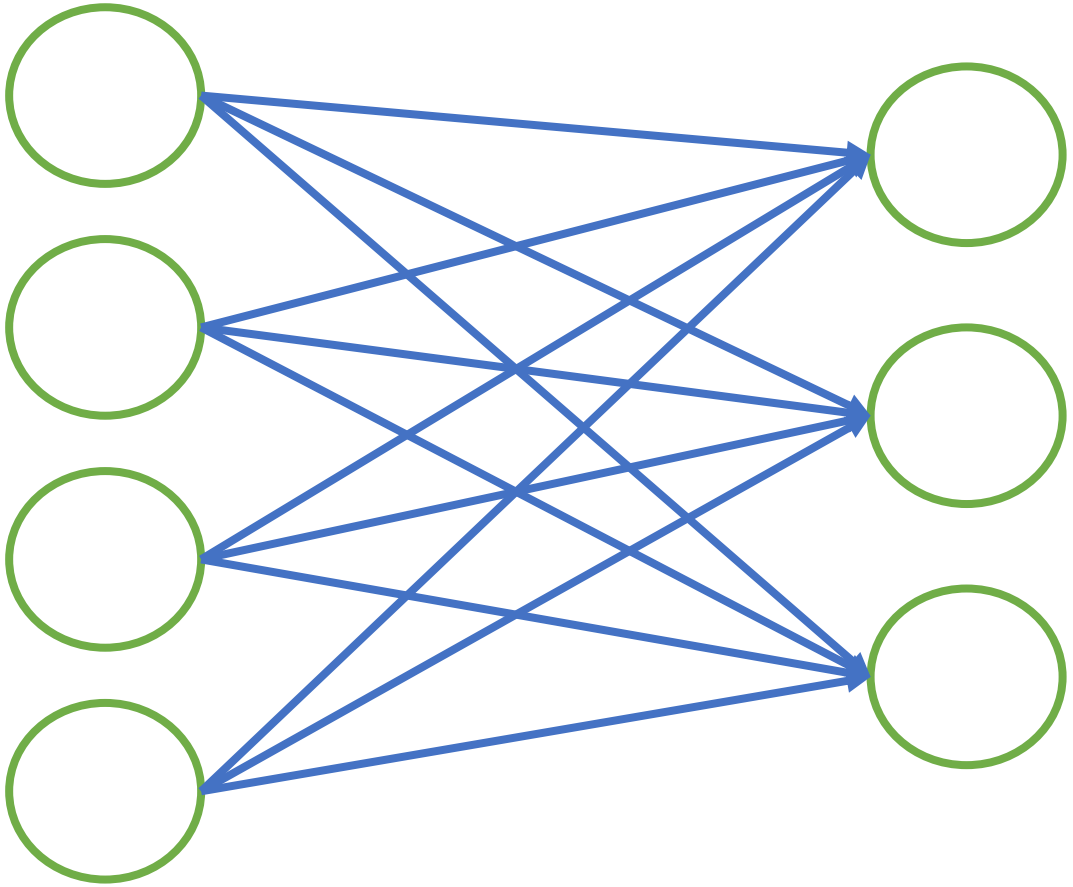
MaxPooling

- 広い範囲の情報を集約する。
- データを小さくして後続の計算量を小さくする。

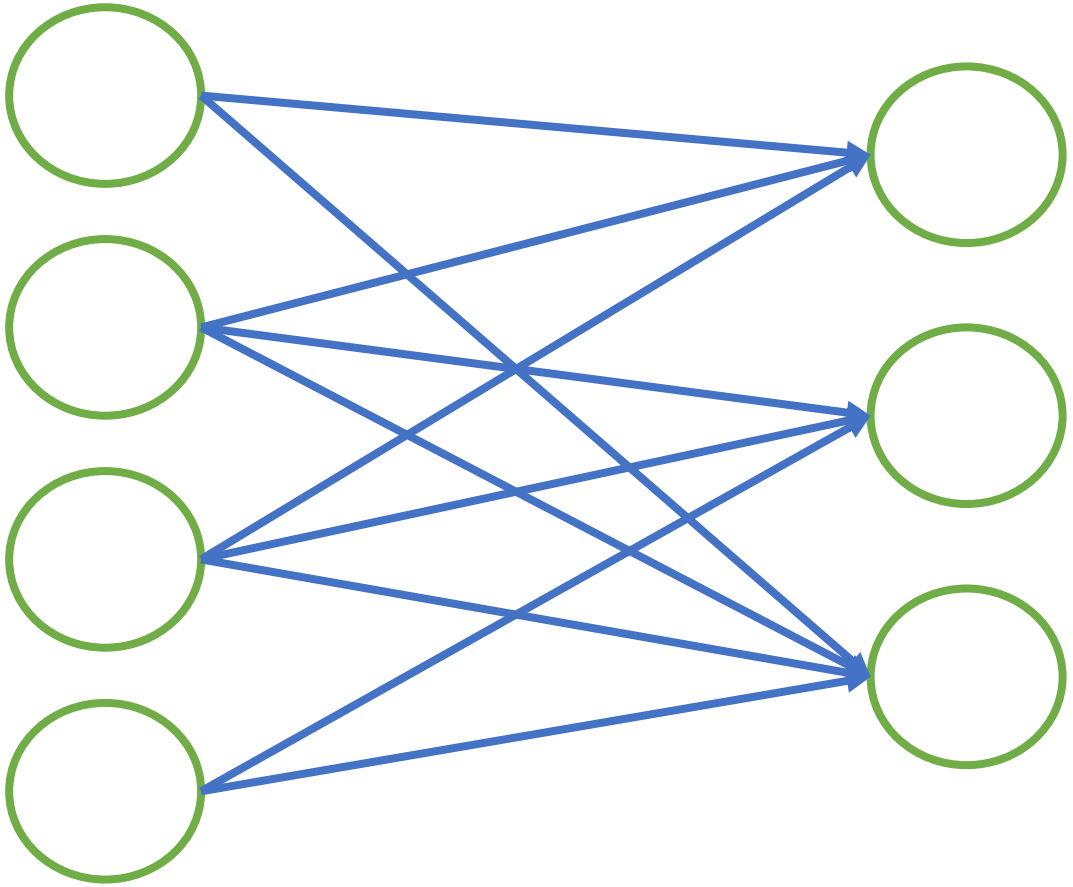
自然言語処理
Dropout

解説動画

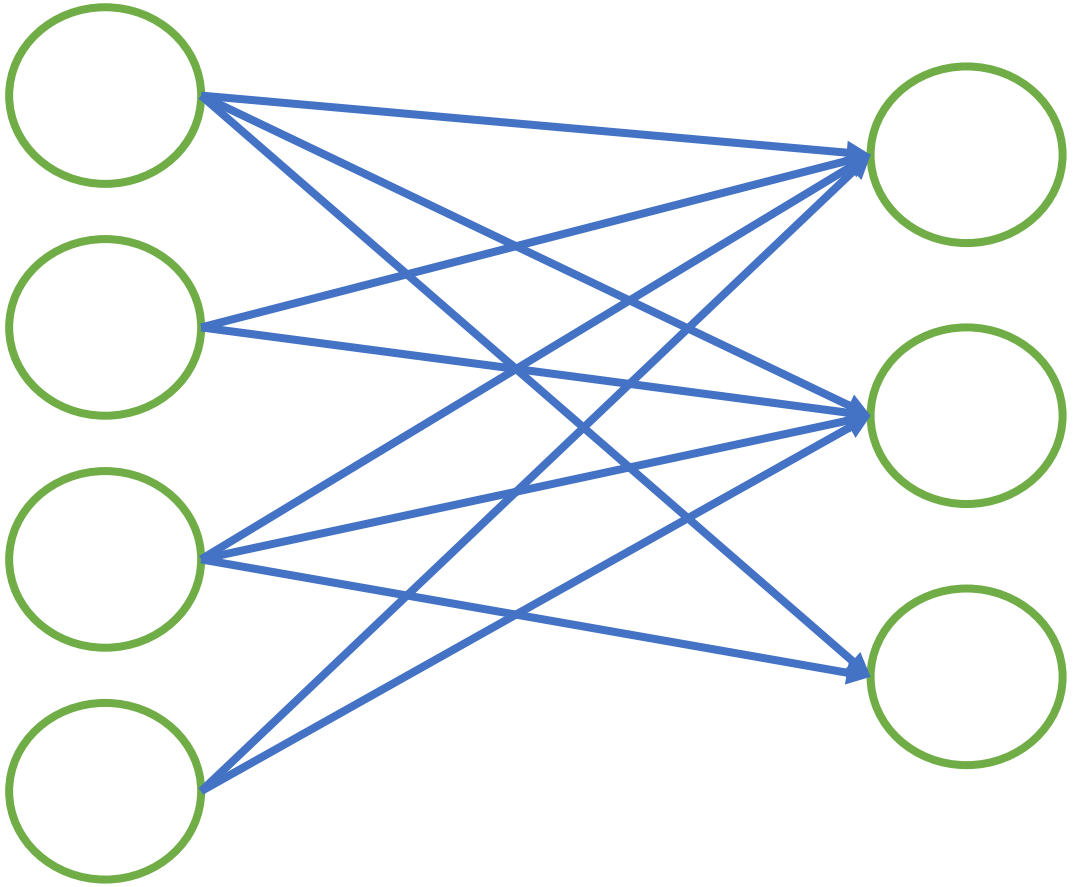
Dropout



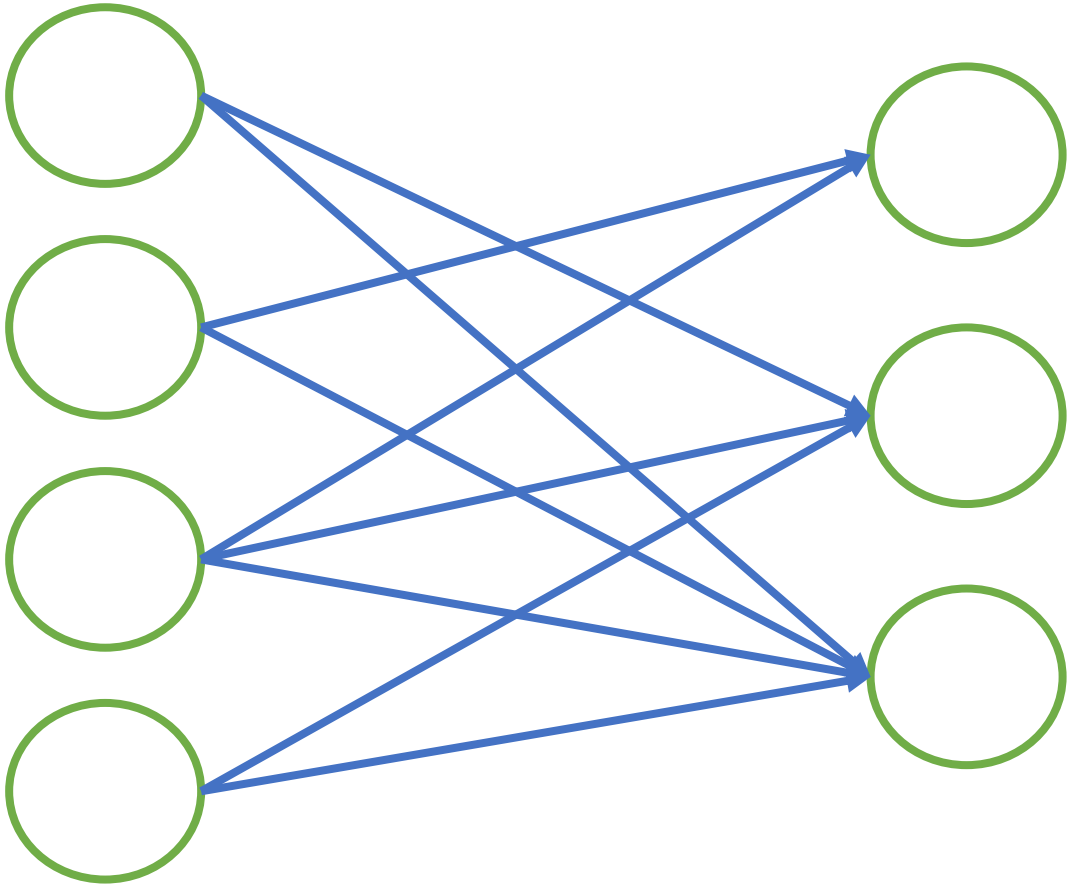
Dropout



Dropout



Dropout



Dropout

仮想的にサブネットワークを構成しそれらの多数決で出力を得る、感じにすることで、般化能力を高める、と言われる（経験的）。

自然言語処理
Batch Normalization

解説動画

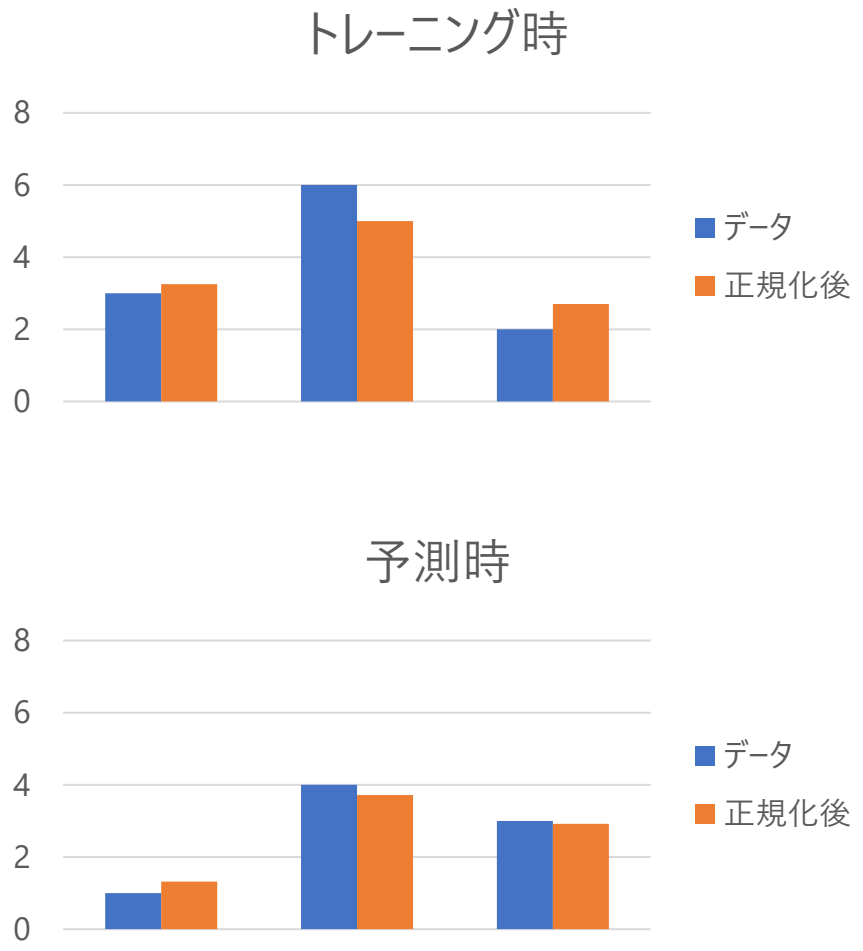
Normalizationとは？

- 入力データや推論中間データが、トレーニング時と予測時とで、分布が大きく異なると困る。
- ある範囲のデータに関して、分布の偏り（凸凹）を小さくする。

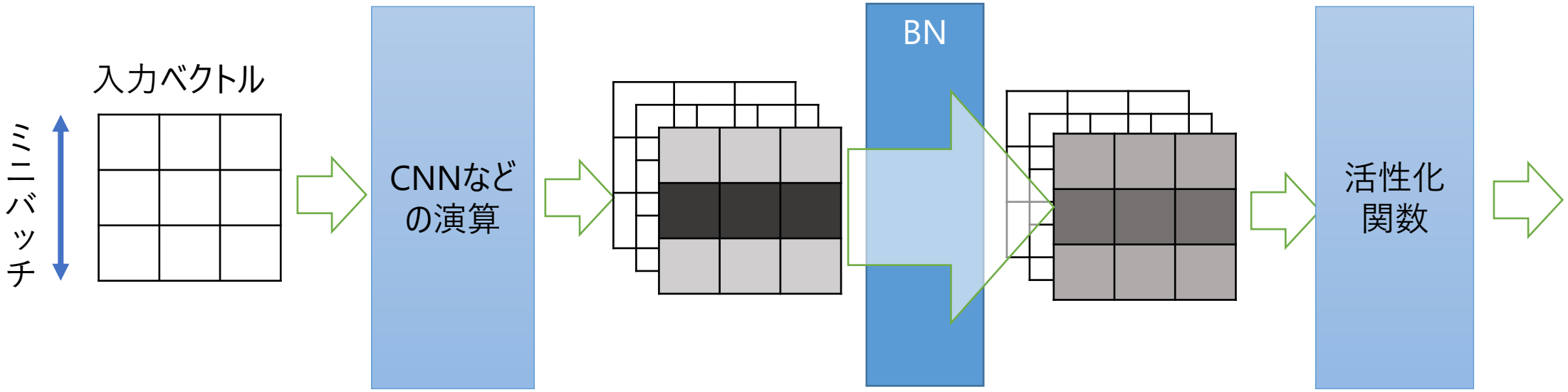
$$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,:},$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2}$$

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}},$$



Batch Normalization

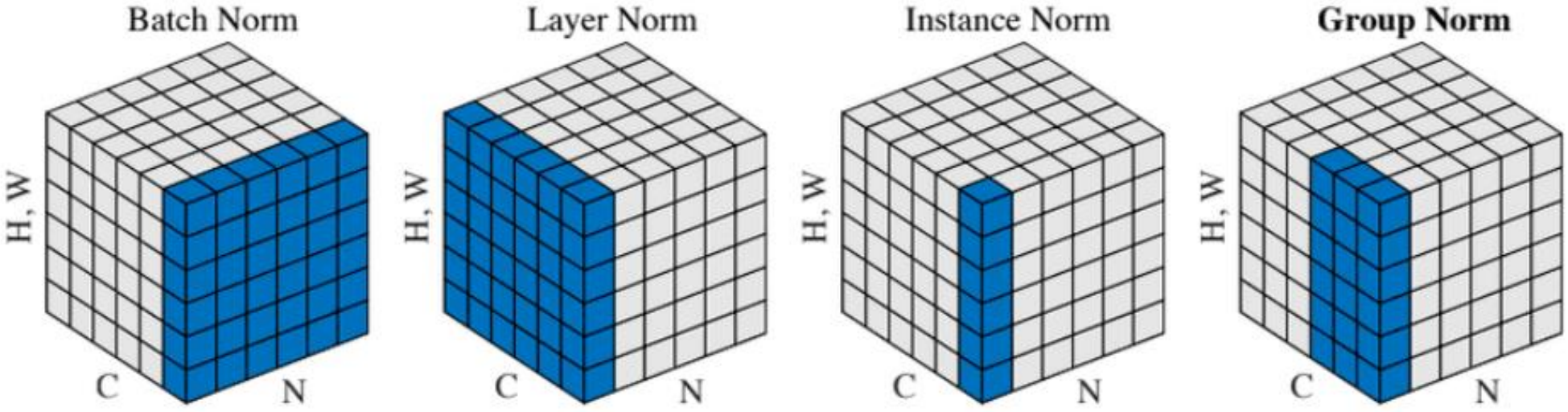


Batch Normalization

- データのばらつきを抑えることで、学習が高速化し、過学習にも効果がある、と言われる。

データの平均と分散をとって補正する

以下、入力データをシリアライズし（H,W）、チャネル（C）、ミニバッチ（N）との3次元立体で示す



ミニバッチ内の正規化

チャネル内の正規化

１入力データ内の正規化

特定のチャネルグループ内の正規化

https://arxiv.org/pdf/1903.10520.pdf

MNISTサンプルコード

MNISTサンプル

- 自然言語処理ではなく、画像処理になりますが、ディープラーニングの典型的な流れと構成要素の概略を把握するために、いいサンプルですので、見ておきます（ニューラルネットの入門書で最終章に用いられていたりする）。
- これは、CNNを利用しています。自然言語処理でもCNNを使うことがあります（例：100本ノック課題８６）。

課題MNISTサンプルの理解

- Mnist_sample.ipynbをダウンロードして、Google Colaboratoryで動かしてみてください。
- 最初のコードは、PyTorchのサンプル集の中の https://github.com/pytorch/examples/tree/master/mnist （もとのPyTorchのコードは、煩雑になりがちなデータ回りの処理もクラス化されているため、Mnistの主要な処理構造が見えやすいものとなっています。）を、教育用に単純化したものです。後続のコード解説、コードのコメント、PyTorchのマニュアルを参照しながら、このコードを咀嚼して下さい。
- そのあとには、参考のため、Convolution結果を表示したり、モデルを変えたコードを入れてあります。

MNistサンプルコードを読むための追加情報

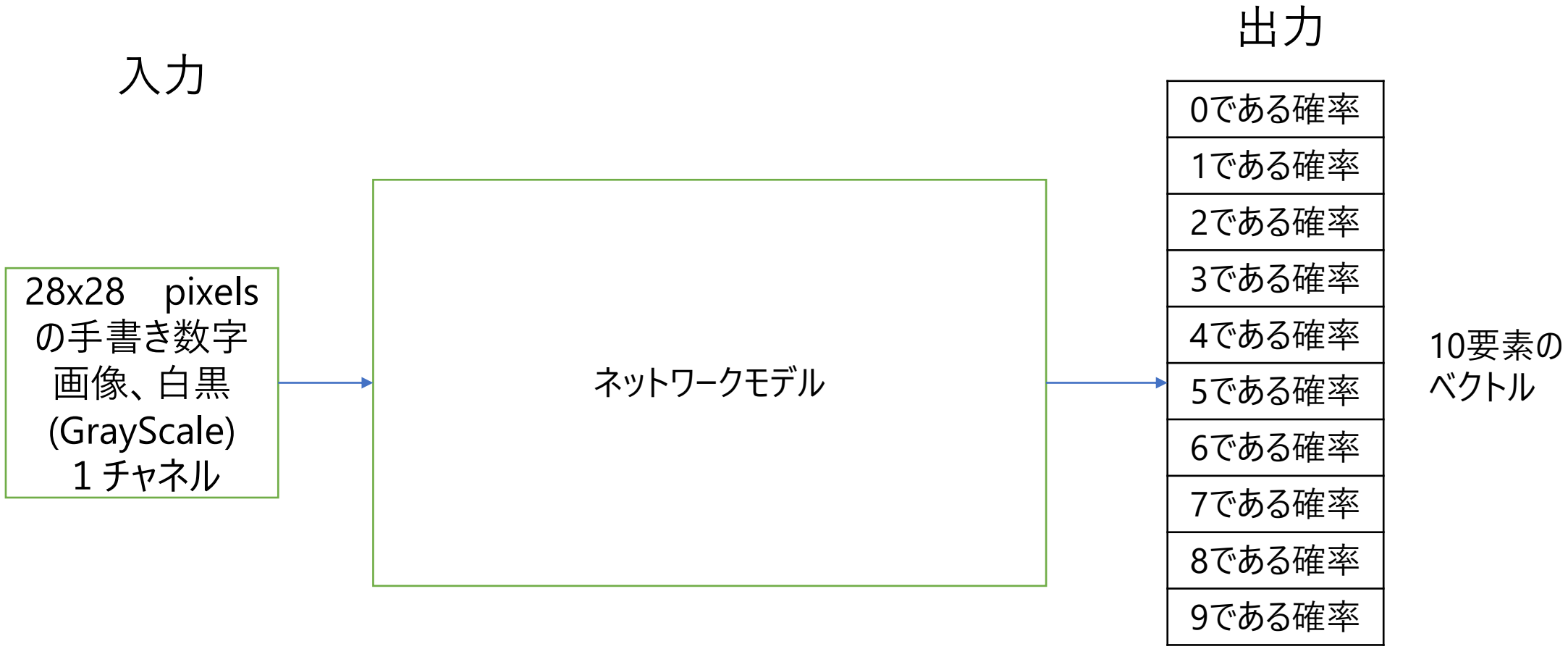
PyTorchマニュアルの読み方

- https://pytorch.org/docs/stable/index.html　へ行ってください。
- 左上に検索ボックスがあるので、そこにわからない書き方をタイプしてください。
- 各ページは英語です。Chromeの場合、該当ページ上で、右クリックで、「日本語に翻訳」を選んでください。

torch.nnとtorch.nn.functionalの違い

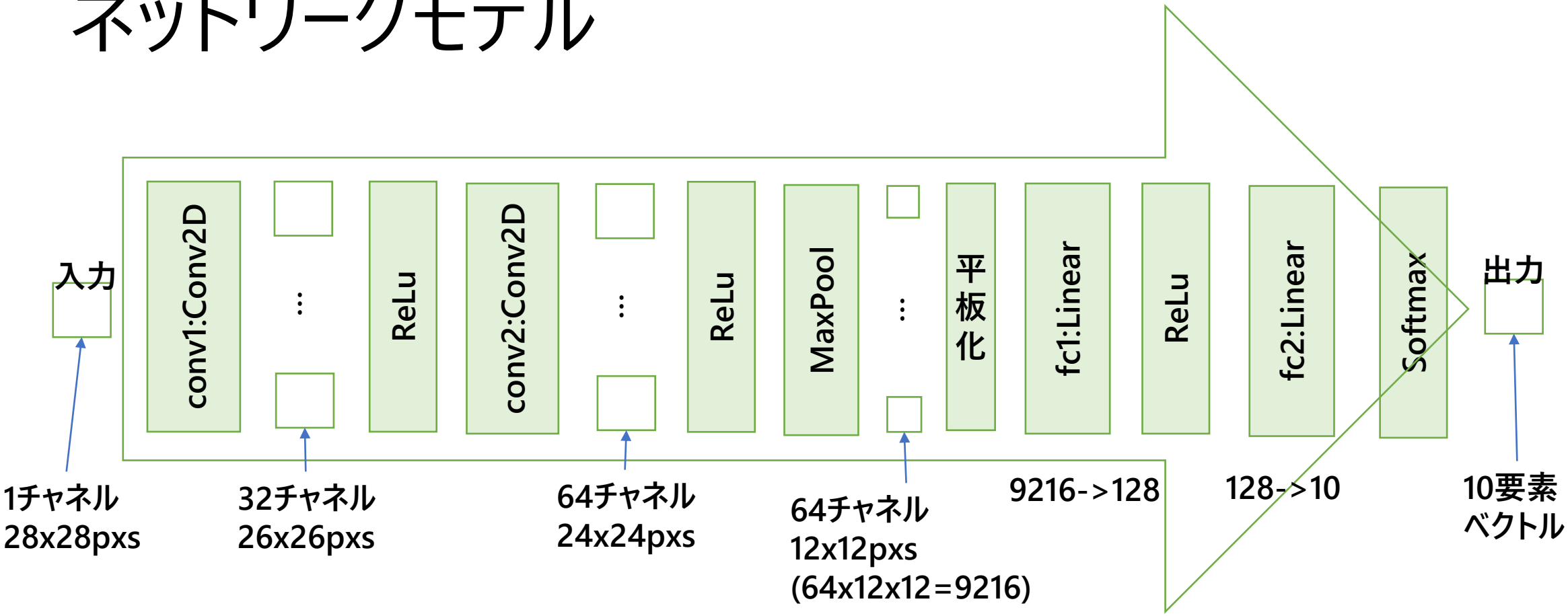
- nn　モジュール
  - クラス定義
  - 例えば、ちょっと複雑な初期設定したい場合、オブジェクトを作っておく。
    - self.conv1=nn.Conv2d(1,32,3,1)
      - nn パッケージ内の　Conv2dクラスのオブジェクトを作成し、初期設定する。
    - x=self.conv1(x)
      - conv1オブジェクトを入力データに対し作用させる。Pythonの呼び出し可能オブジェクトの機能を使い、__call__している。
- functional　モジュール
  - nn　クラスの機能を、関数として利用できるようにしたもの。
  - 例えば
    - x=F.relu(x)
    - x=F.max_pool2d(x, 2)
      - nn　モジュールの MaxPool2dのオブジェクトを作成し、引数で初期化して、入力データに作用させる。

Mnist サンプルの入出力

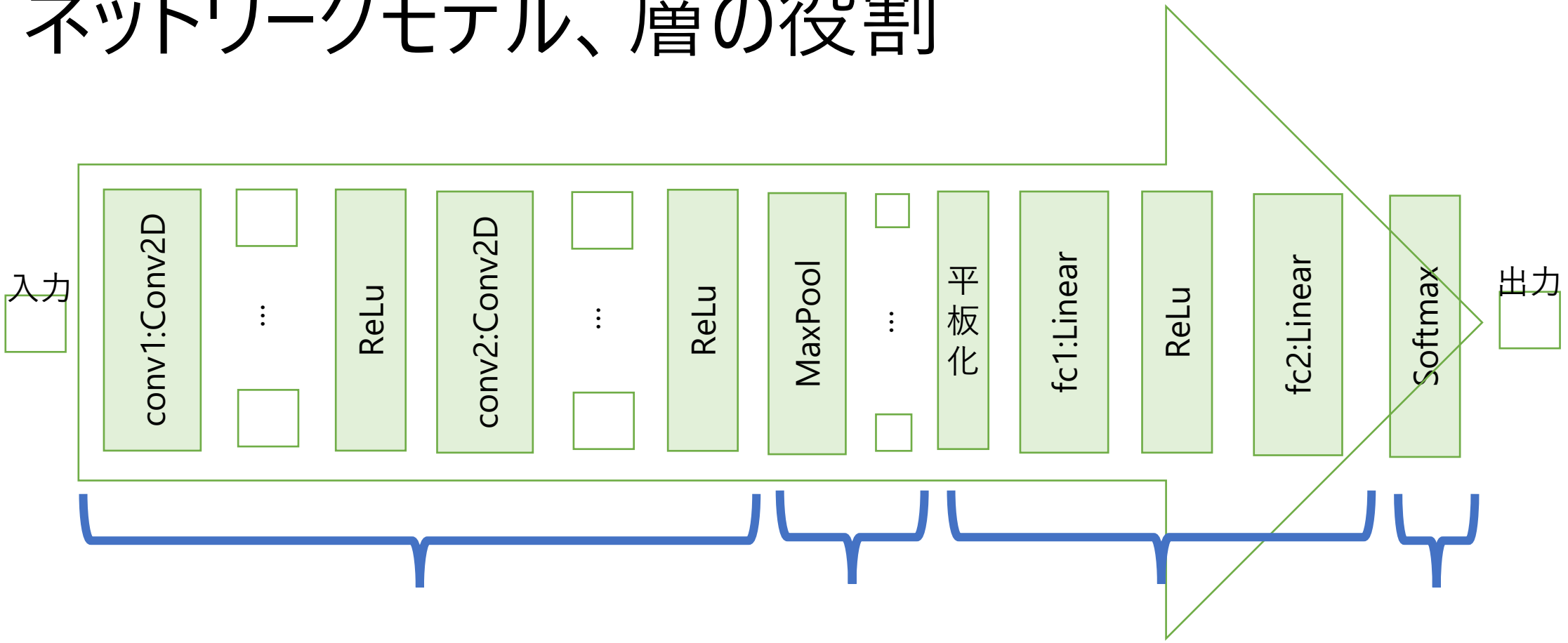


Train/Testは、データローダーを介して
64個ずつのミニバッチ単位で計算

ネットワークモデル


入力
conv1:Conv2D
ReLu
conv2:Conv2D
ReLu
MaxPool
平板化
fc1:Linear
ReLu
fc2:Linear
Softmax
出力
1チャネル
28x28pxs
32チャネル
26x26pxs
64チャネル
24x24pxs
64チャネル
12x12pxs
(64x12x12=9216)
9216->128
128->10
10要素
ベクトル

ネットワークモデル、層の役割



CNN2層で画像の何らかの特徴を抽出

サイズを小さくし、グローバルに特徴をまとめる

全結合2層で情報の取捨選択・集約

確率に変換

Netクラス__Init__

nn.Conv2d(1, 32, 3, 1)

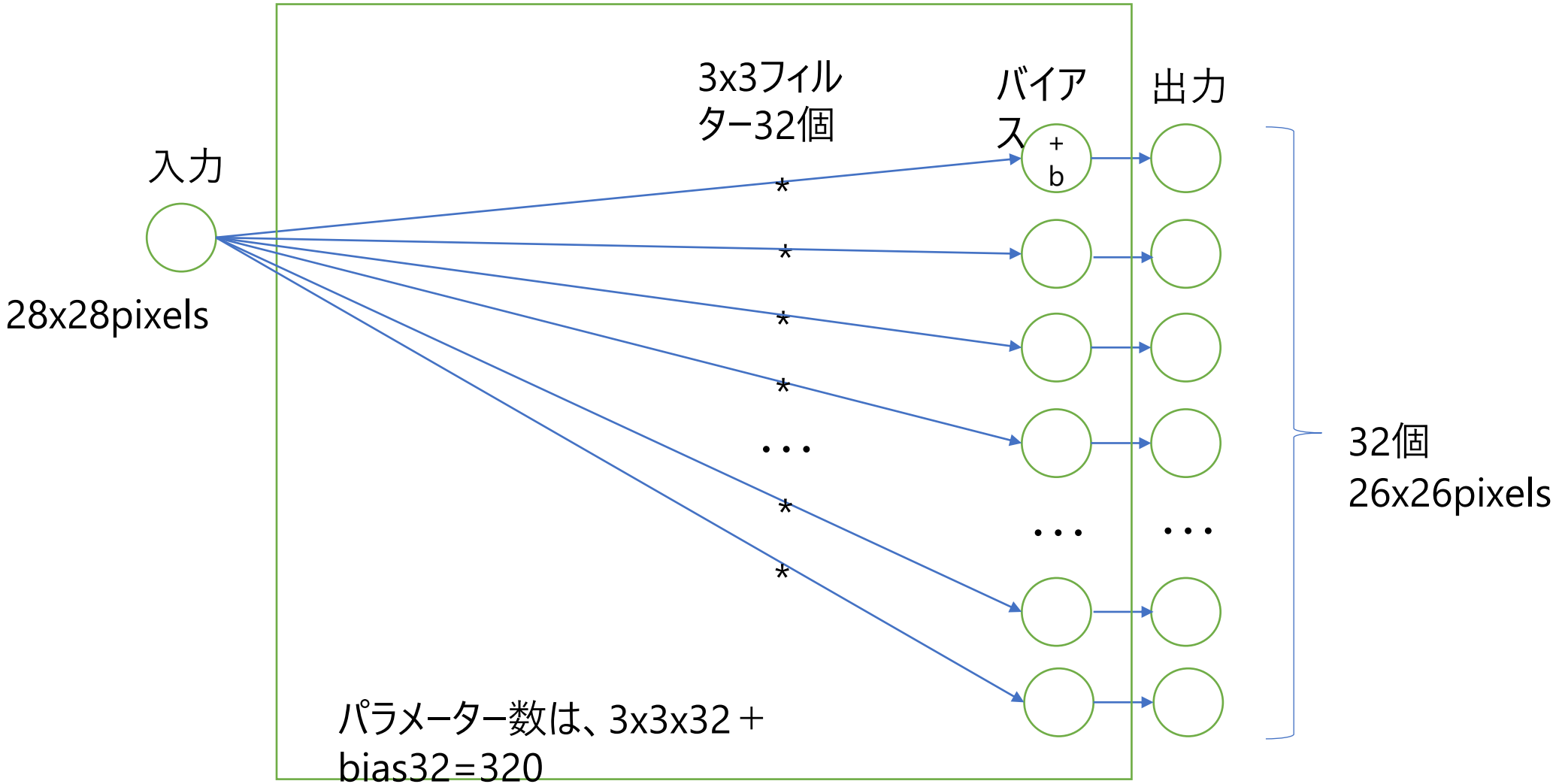
入力チャネル数

出力チャネル数

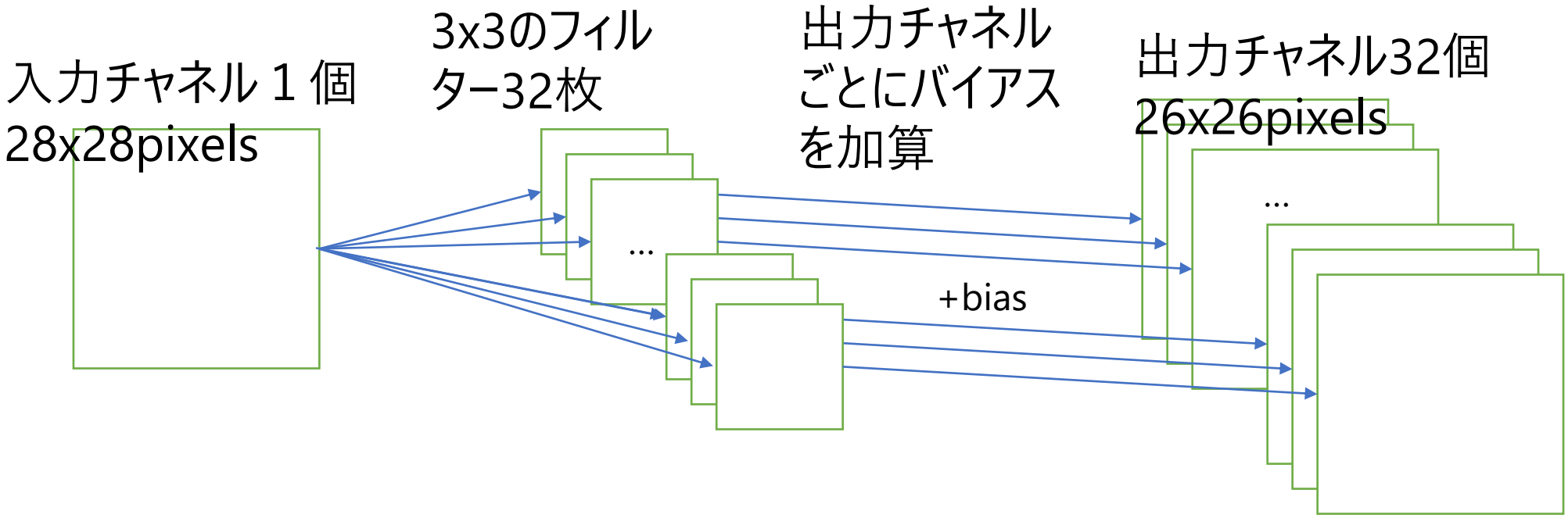
カーネルサイズ：整数か、
(高さ、幅)のTuple

ストライド：フィルターの窓
を移動させるときの移動数

conv1

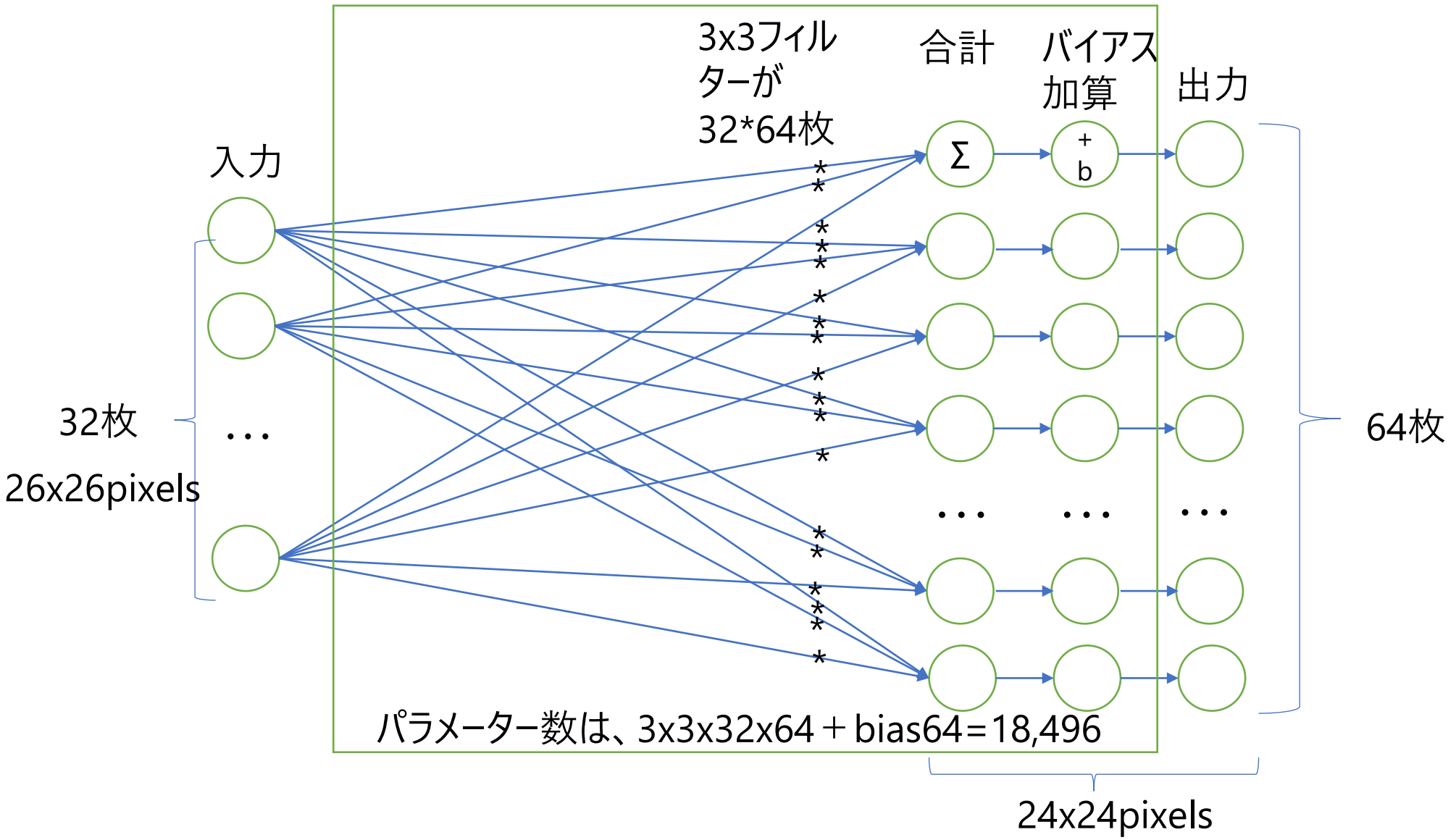

3x3フィルター32個
バイアス
出力
入力
+
b
28x28pixels
32個
26x26pixels
パラメーター数は、3x3x32＋bias32=320

conv1



パラメーター数は、3x3x32＋bias32=320

conv2


3x3フィルターが32*64枚
合計
バイアス加算
出力
入力
Σ
+ b
32枚
26x26pixels
64枚
パラメーター数は、3x3x32x64＋bias64=18,496
24x24pixels

nn.Conv2d(32, 64, 3, 1)

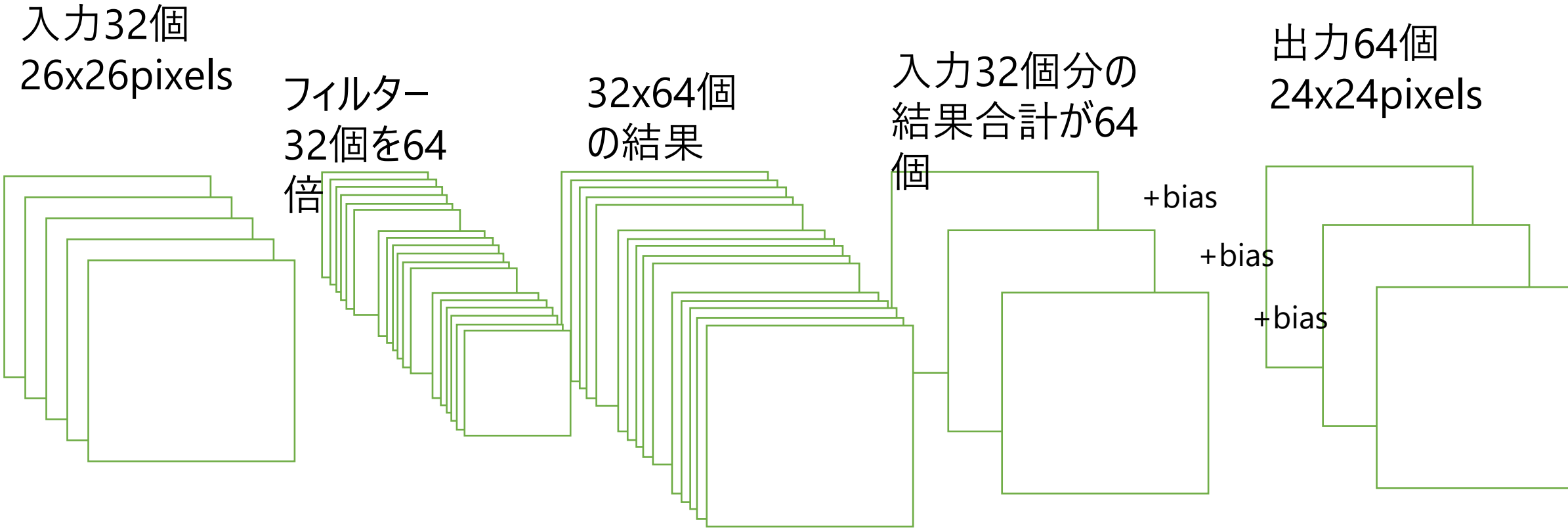
入力チャネル数

出力チャネル数

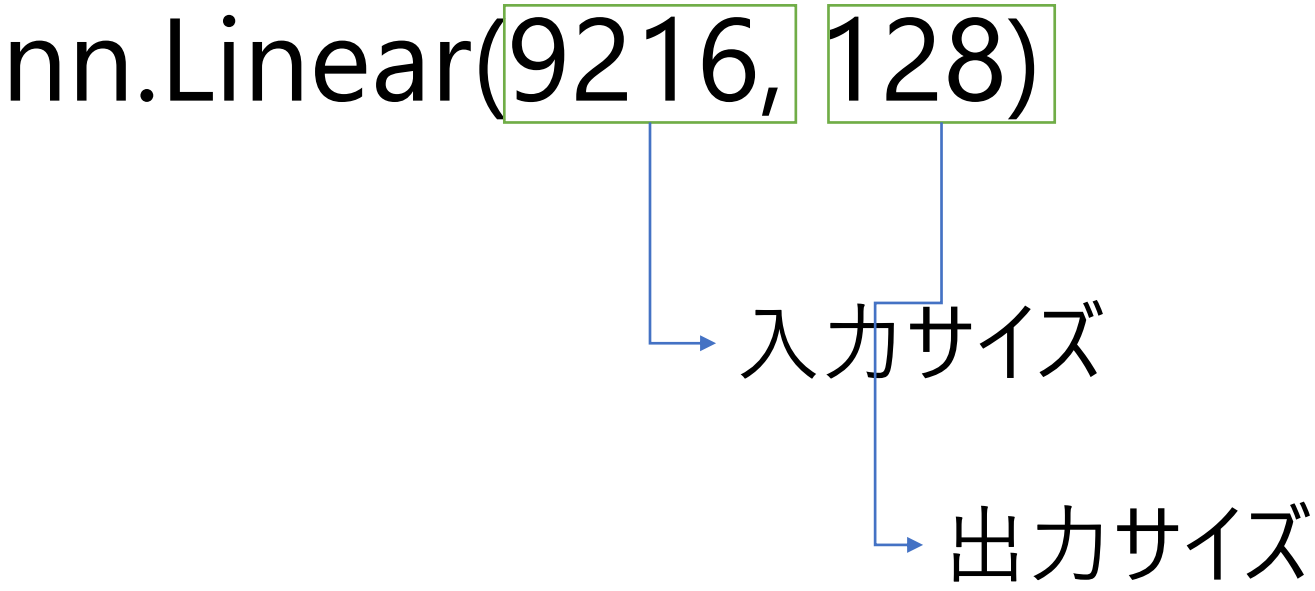
カーネルサイズ：整数か、
(高さ、幅)のTuple

ストライド：フィルターの窓
を移動させるときの移動数

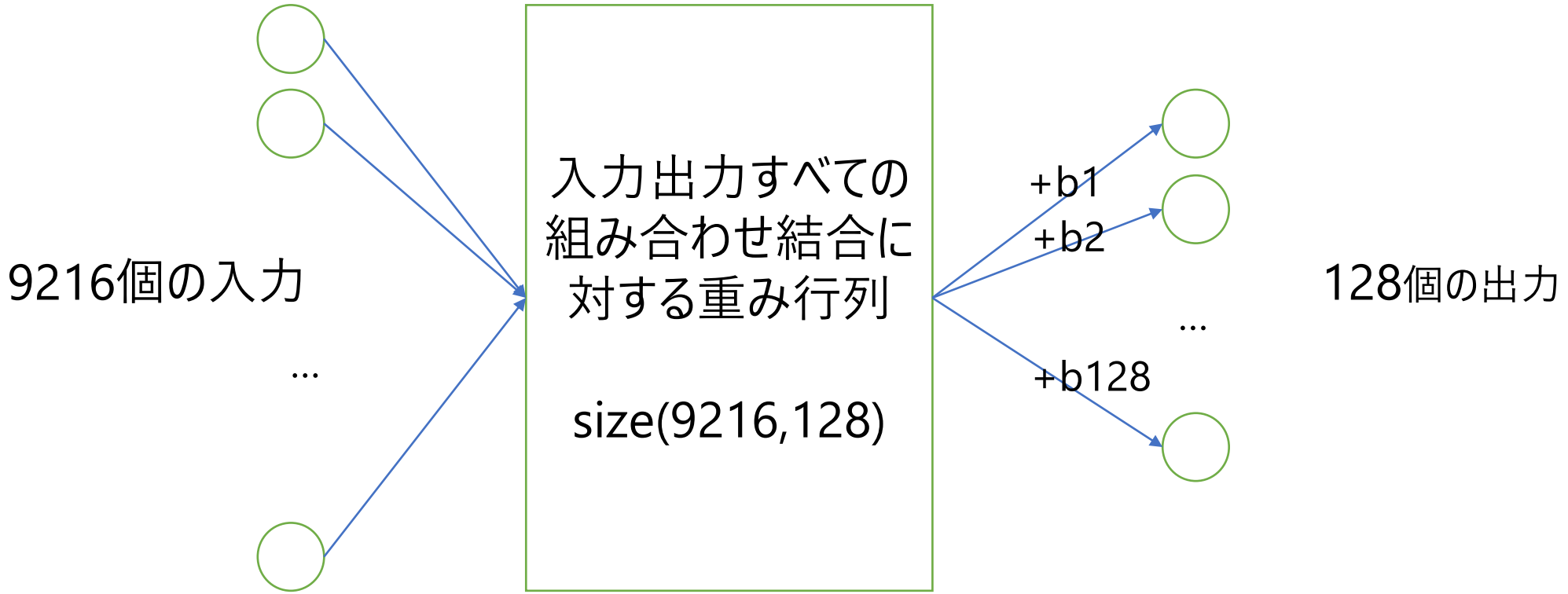
conv2



パラメーター数は、3x3x32x64＋bias64=18,496

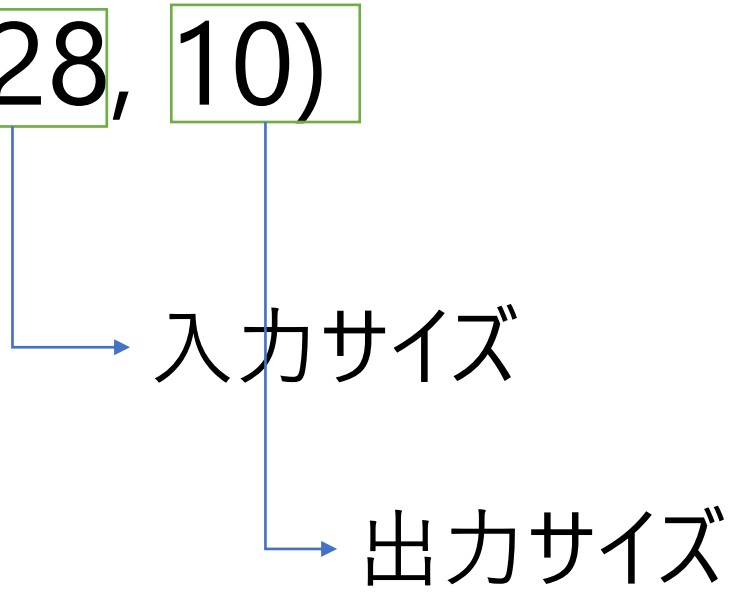

nn.Linear(9216, 128)
入力サイズ
出力サイズ

fc1

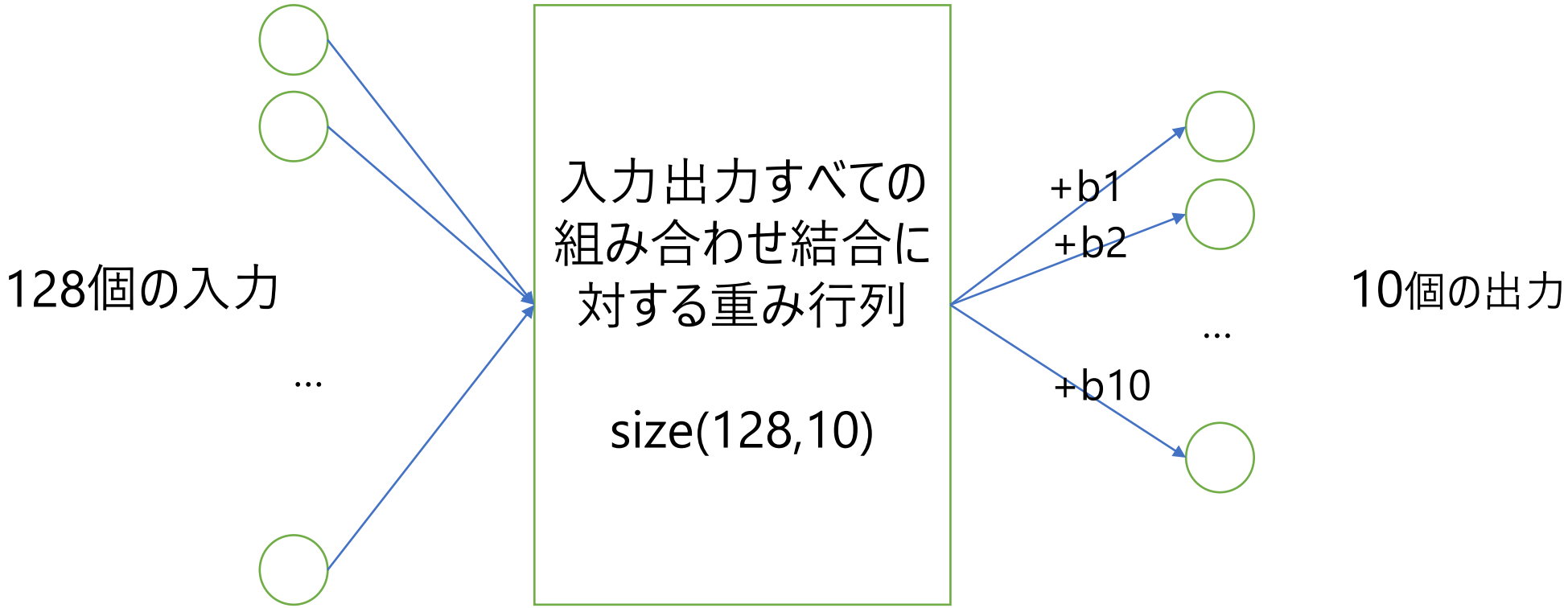


実際には、ミニバッチの数分、まとめて計算され、
入力も出力も、1次元ベクトルではなく、ミニバッチを行方向に収めた行列となる

nn.Linear(128, 10)



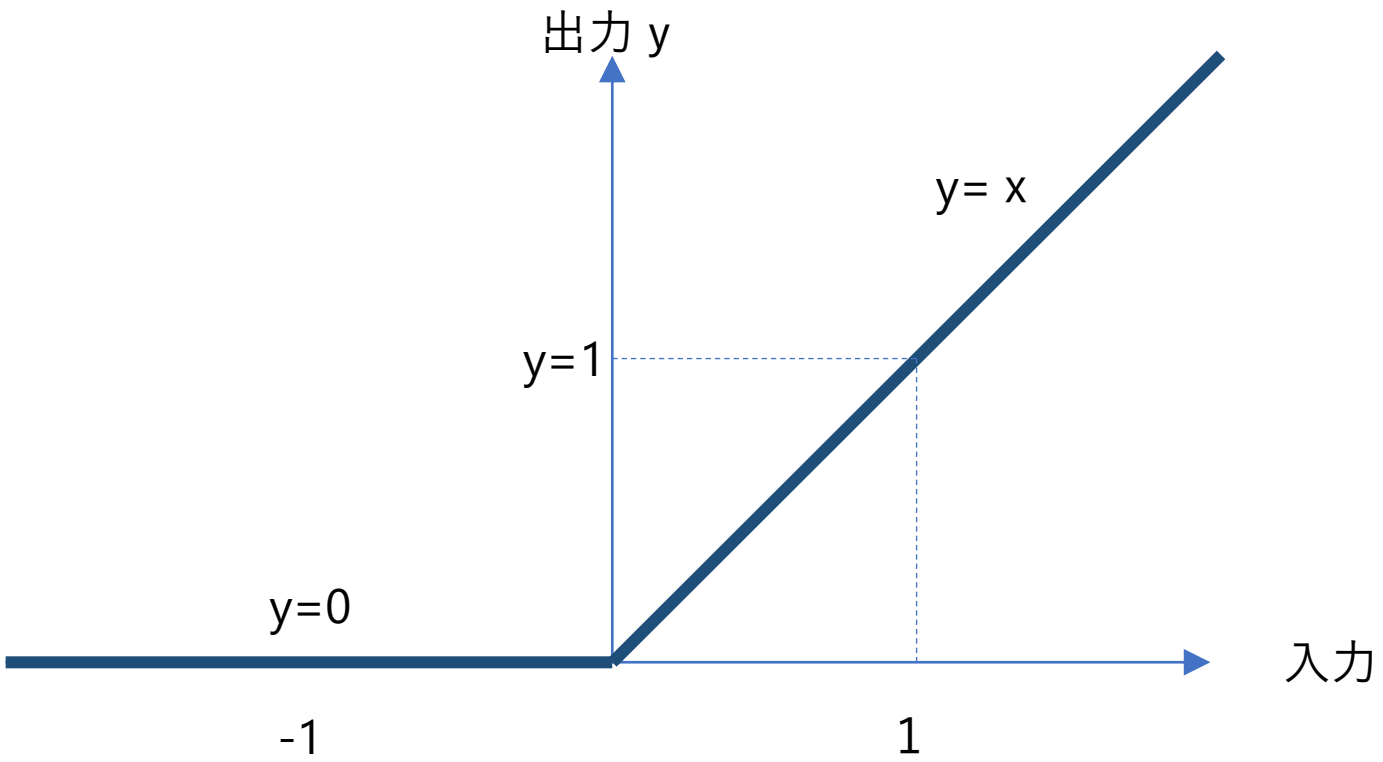
fc2



実際には、ミニバッチの数分、まとめて計算され、
入力も出力も、1次元ベクトルではなく、ミニバッチを行方向に収めた行列となる

Netクラス　forward

活性化関数ReLU(Rectified Linear Unit)



max_pool2D(x, 2)

入力：（ミニバッチ数、チャネル数、高さ、幅）の
sizeを持つテンソル

カーネルサイズ：整数
省略可能引数にストライド（ウィンドウの移動幅）があり、
ストライドのデフォルトはカーネルサイズ

MaxPooling：広い範囲の情報を集約する。データを小さくして計算量を小さくする。

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

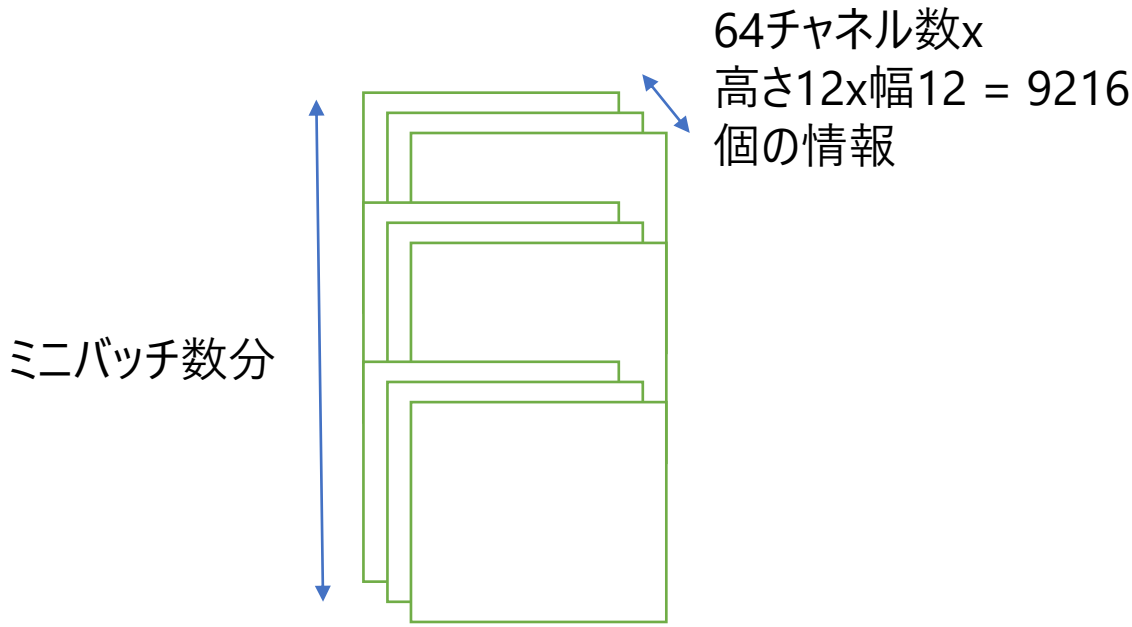
7	9
7	9

入力

出力

torch.flatten(x, 1)

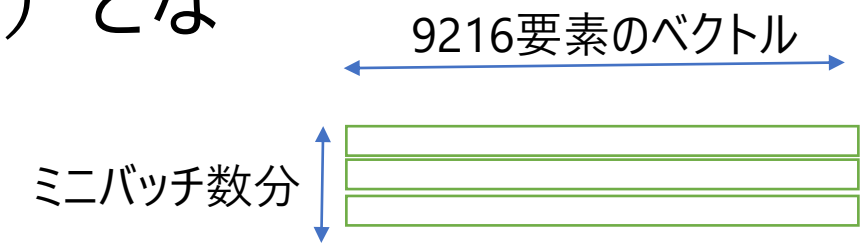
入力テンソル：ここでは、sizeは (ミニバッチ数、チャネル数、高さ、幅）



平板化する最初の次元１　-> size が (ミニバッチ数、平板化されたデータ）となる。

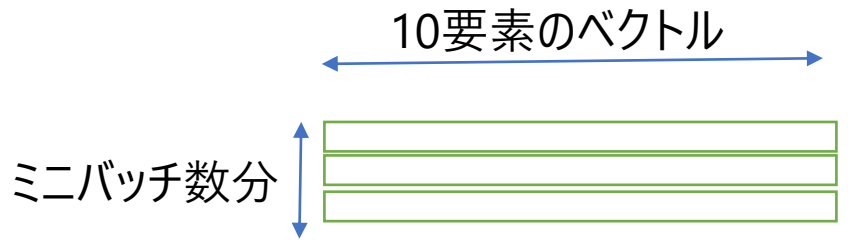
tensor([[...],[...],....,[...]]

ミニバッチの数だけ

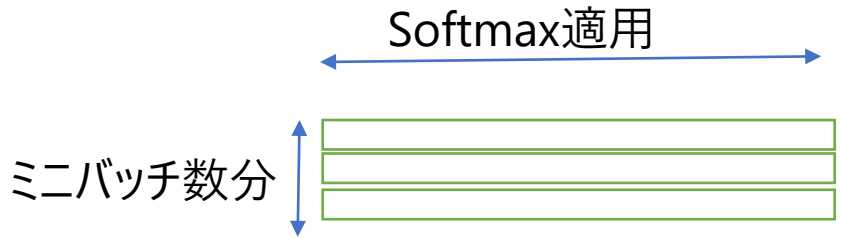


F.softmax(x, dim=1)

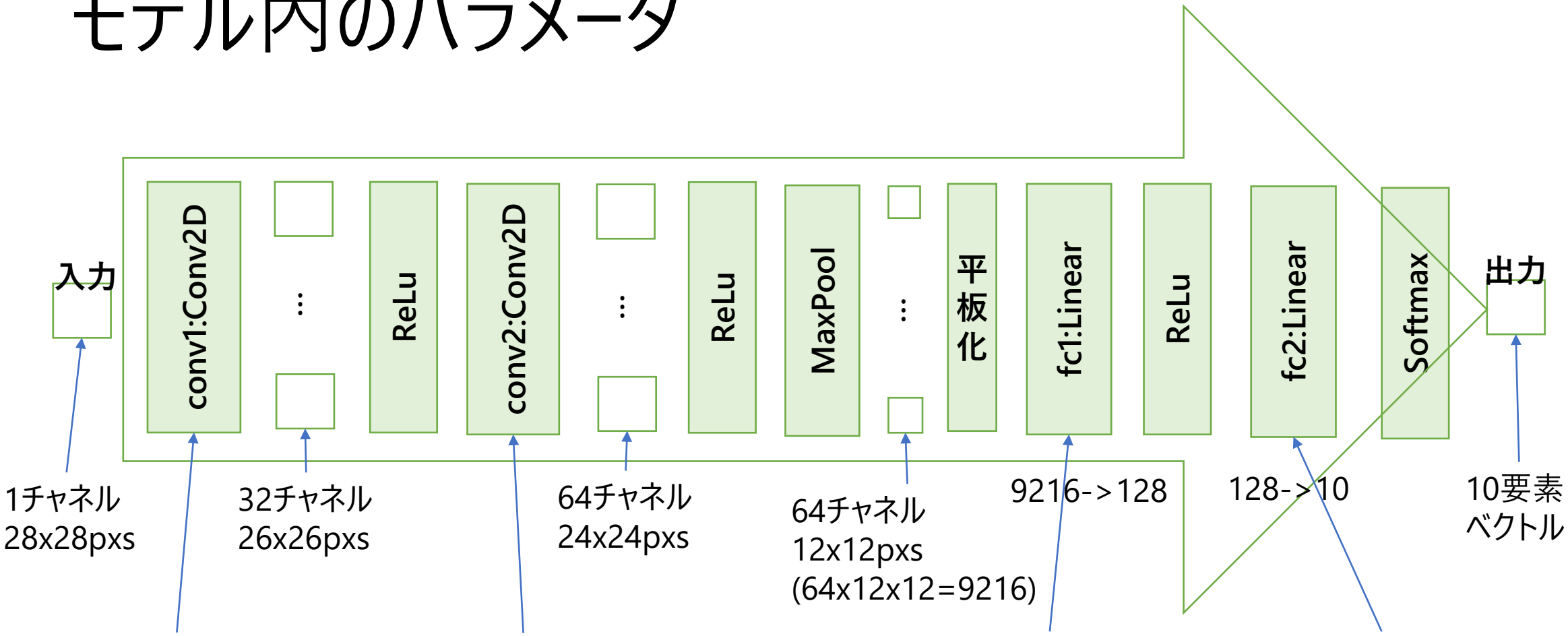
入力



softmaxをとる次元：0がミニバッチ、1 が10要素のベクトル



モデル内のパラメータ


入力
conv1:Conv2D
ReLu
conv2:Conv2D
ReLu
MaxPool
平板化
fc1:Linear
ReLu
fc2:Linear
Softmax
出力
1チャネル
28x28pxs
32チャネル
26x26pxs
64チャネル
24x24pxs
64チャネル
12x12pxs
(64x12x12=9216)
9216->128
128->10
10要素
ベクトル
パラメーター数
3x3x32＋bias32
=320
パラメーター数
3x3x32x64＋bias64
=18,496
パラメーター数
9216x128+bias128
=1,179,776
パラメーター数
128x10+bias10
=1,290

main

データセット、データローダー

- データセット
  - 入力データを、Pytorch Tensor形式に変換したり、値を正規化したり、など変換してから、送る。
- データローダー
  - ミニバッチ単位にまとめて、モデルに送る。

optim.SGD(model.parameters(), lr=0.1)

モデルのパラメータ

学習レート

SGD (Stochastic Gradient Decent) 確率的勾配降下法。ほかにも。

勾配降下法なのだが、入力データはシャッフルしてあってランダムなので、確率的。

SGD以外にいろんなOptimizerが準備されている。

StepLR(optimizer, step_size=1, gamma=0.7)

エポック何個で学習具合を低減させるか

学習レートの逓減率

エポックが変わったときに、段階的に、一律に学習レートを低減させる。
ほかに低減させるやり方のOptionが何個か準備されている。

train, test

ミニバッチ

$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = (x1w11 + x2w21 + x3w31, x1w12 + x2w22 + x3w32)$$

データを何個分かまとめる

ミニバッチ
3個

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix}\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

GPUの能力活用

outputs = model(inputs) が、forward呼び出しに化ける訳

Pytorchは、Pythonの呼び出し可能オブジェクトの機能を多用している

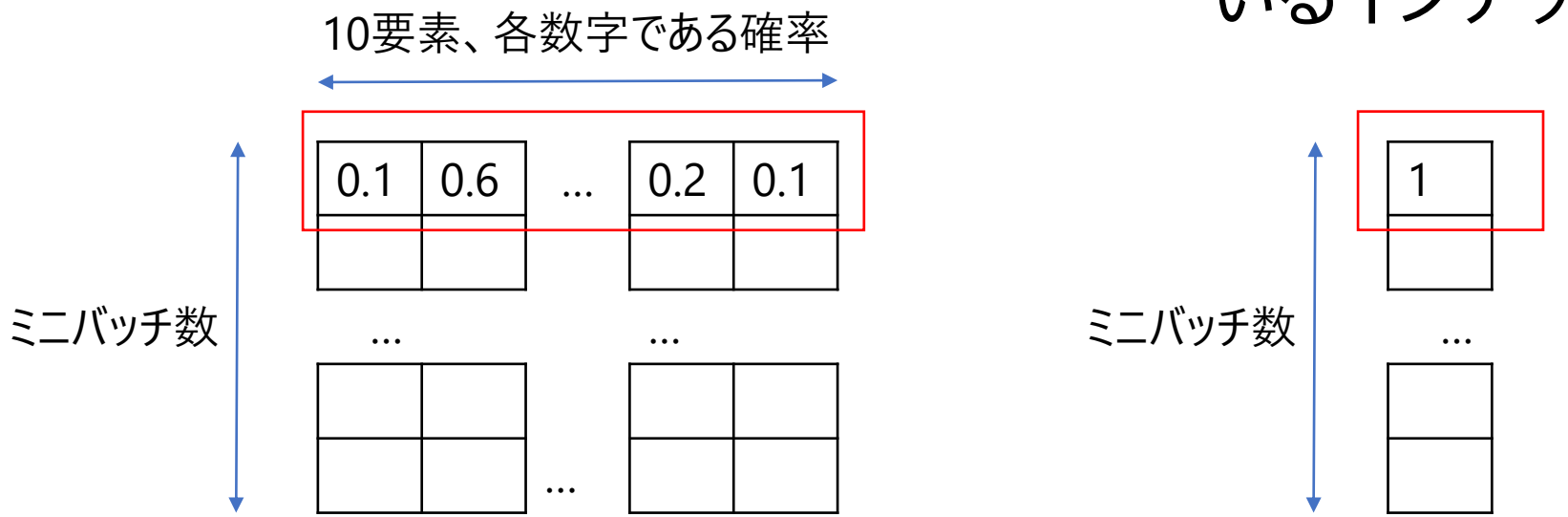
- object.__call__(self[, args...])
  - Called when the instance is "called" as a function; if this method is defined, x(arg1, arg2, ...) roughly translates to type(x).__call__(x, arg1, ...).
  - オブジェクト()　と書くと、クラス.__call__(インスタンス、引数...)　に化ける

- Pytorchのコード
  - moduleクラスの　__call__　メソッドは、中で forward 呼び出しを行っている。

F.cross_entropy(output, target)

検査するデータ

正解ラベル 1 が立っているインデックス0,..,9



赤の場合、出力は、-log(0.6)

発展：softmax の結果0..1の値でロスを計算するより、0..1の各値の対数とって-∞..0にしてから、ロス計算に回したほうが、分解能が高い。

- Net
  - forward
    - …
    - output = F.log_softmax(x, dim=1)
    - …
- Train/Test
  - …
  - loss = F.nll_loss(output, target)
  - …

発展：計算結果が毎回変わる要素を減らすため、乱数発生器の初期値を固定してもいい。

- main
  - …
  - torch.manual_seed(args.seed)
  - …

発展：過学習しないような対策

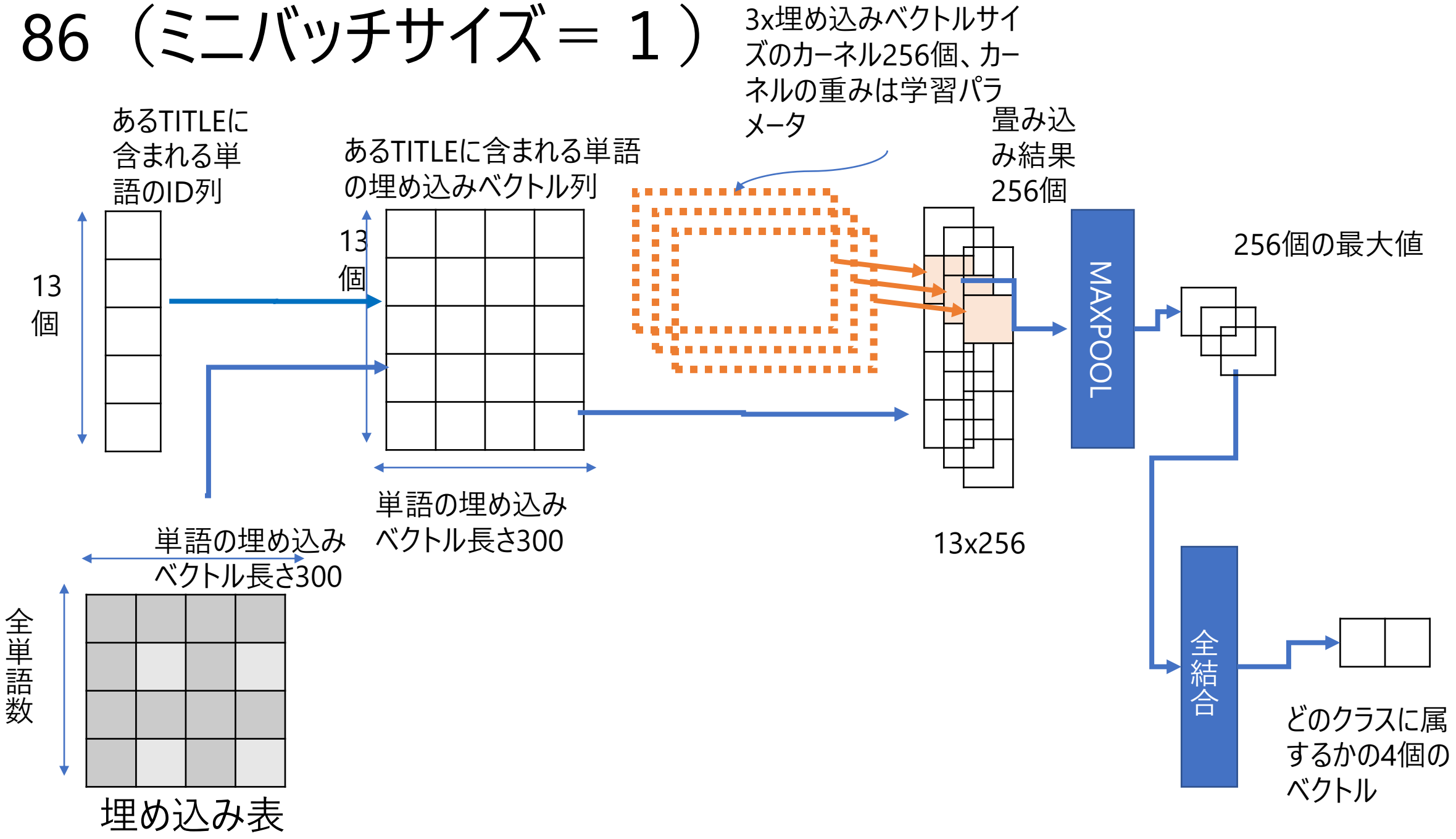
- Net
  - __init__
    - …
    - self.dropout1 = nn.Dropout2d(0.25)
    - self.dropout2 = nn.Dropout2d(0.5)
    - …
  - forward
    - …
    - x = self.dropout1(F.max_pool2d(F.relu(self.conv2(x)))
    - …
    - x = self.dropout2(F.relu(self.fc1(x)))

- モデルにランダムに欠落ノードを作ることで、常時接続した単一のモデルにはない汎用性が仕込まれるらしい。経験的な知見でしかない。

100本ノック第9章課題80,86

- 「100本ノック」の9章の課題 の80(データ準備と86/87がCNNに関する課題です。
- 「NLP、CNNRNNTransformer.ipynb」というノートをコピーして下さい。80、86回答例コードを完成し、実行ログを残してください。

86（ミニバッチサイズ＝１）



確認クイズ

- 確認クイズをやってください。