PythonとKerasによるディープラーニング

第5章 コンピュータビジョンのためのディープラーニング

5.1 畳み込みニューラルネットワークの紹介

16T4063F 結城洸太

5.1 畳み込みニューラルネットワークの紹介

- 畳み込みニューラルネットワーク(CNN)とはなにか

→コンピュータビジョンのタスクに最適な機械学習モデル

  - 視覚的な分類問題に最適。
  - 学習する表現が解釈しやすい。

CNNの例を実践的な角度から見てみる。(MINISTを例に)

(28×28ピクセル)

5.1 畳み込みニューラルネットワークの紹介

- 基本的なCNNのインスタンス化

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers. Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers. Conv2D(64, (3, 3), activation='relu'))
```

5.1 畳み込みニューラルネットワークの紹介

- CNNの上に全結合分類機(Dense層のスタック)を追加

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

5.1 畳み込みニューラルネットワークの紹介

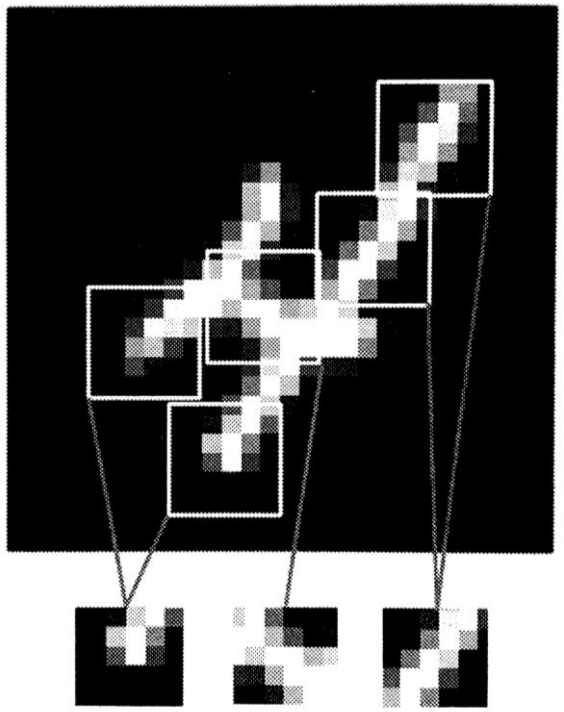
- アーキテクチャを表示してみる。

>>> **model.summary()**

Layer(type)	**Output shape**	**Param #**
Conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
Maxpooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
Conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
Maxpooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
Conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
Flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

5.1.1 畳み込み演算

- 全結合層の場合
  全てのピクセルを含む大域的なパターンを学習する。

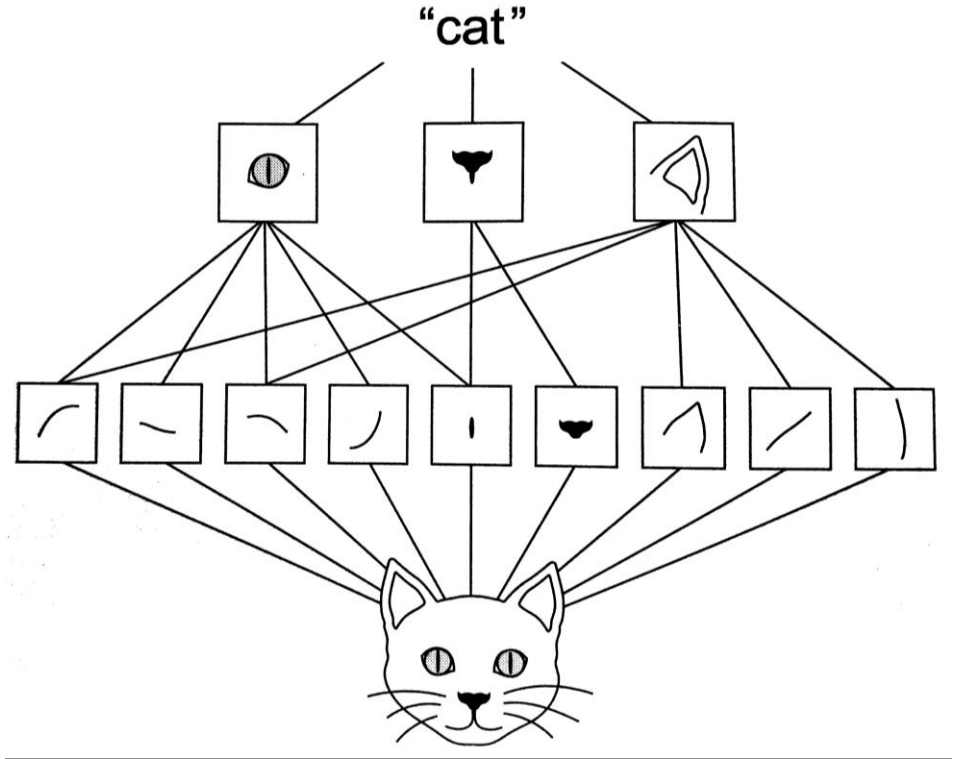


- 畳み込み層の場合
  局所的なパターンを学習する。

5.1.1 畳み込み演算

CNNの特徴

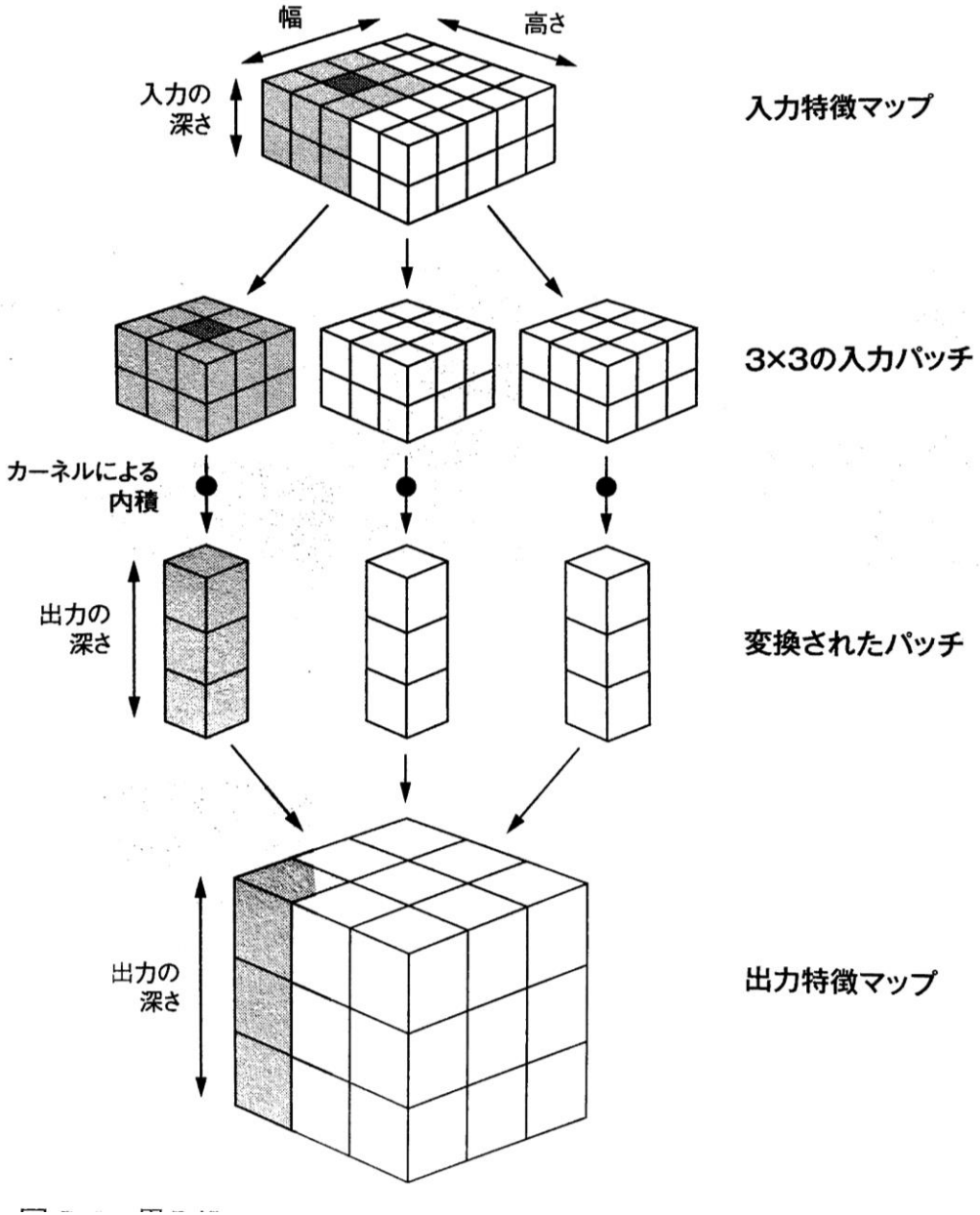
- CNNが学習するパターンは移動不変である。
- CNNはパターンの空間階層を学習できる。



5.1.1 畳み込み演算

畳み込み演算は、**特徴マップ**に対して実行され、**出力特徴マップ**を生成する。

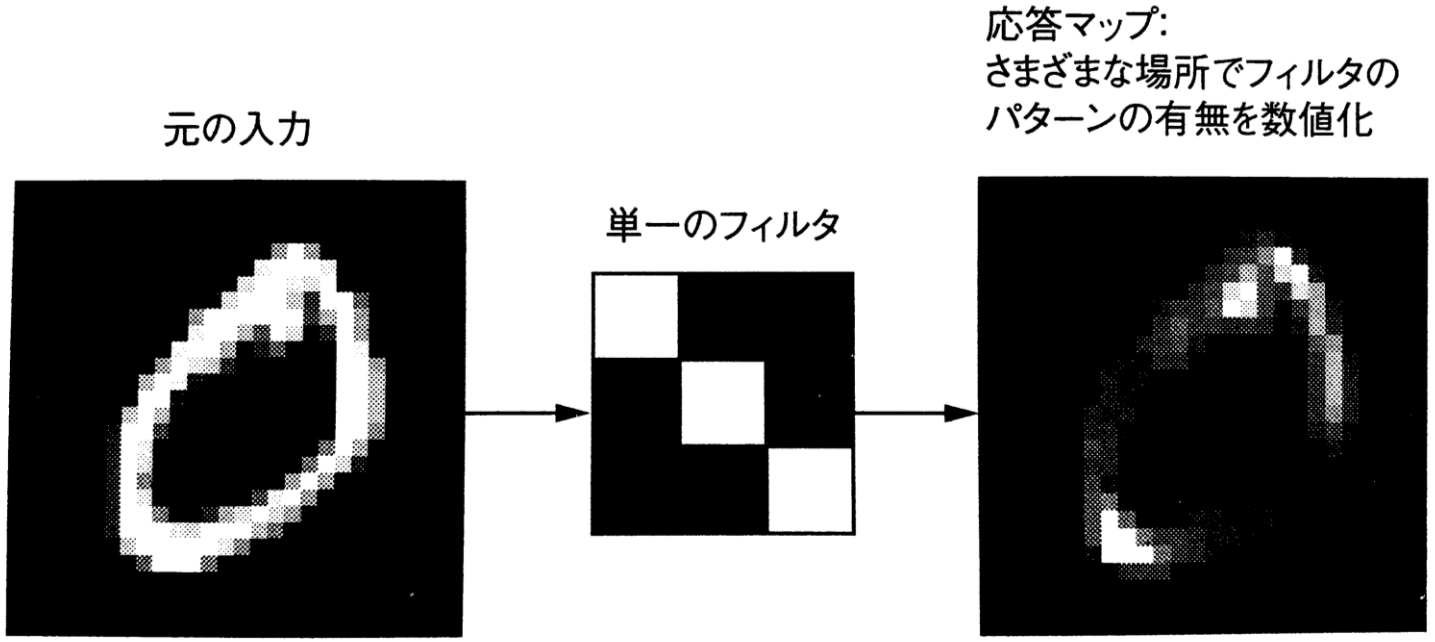
- 特徴マップ
  - 2つの空間軸(幅と高さ)と1つの深さ軸(チャネル軸)に基づく3次元テンソル



5.1.1 畳み込み演算

最初畳み込み層のサイズ
入力(28, 28, **1**) → 出力(26, 26, **32**)

入力に対して32個のフィルタを計算する



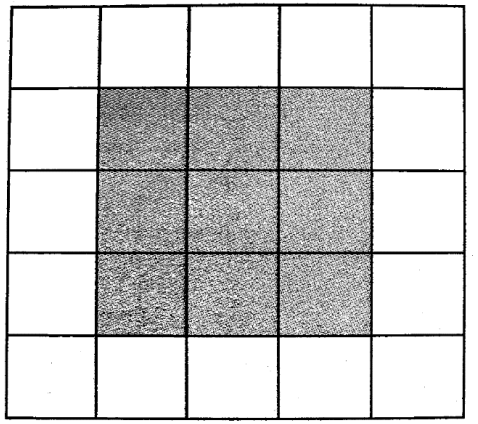
5.1.1 畳み込み演算

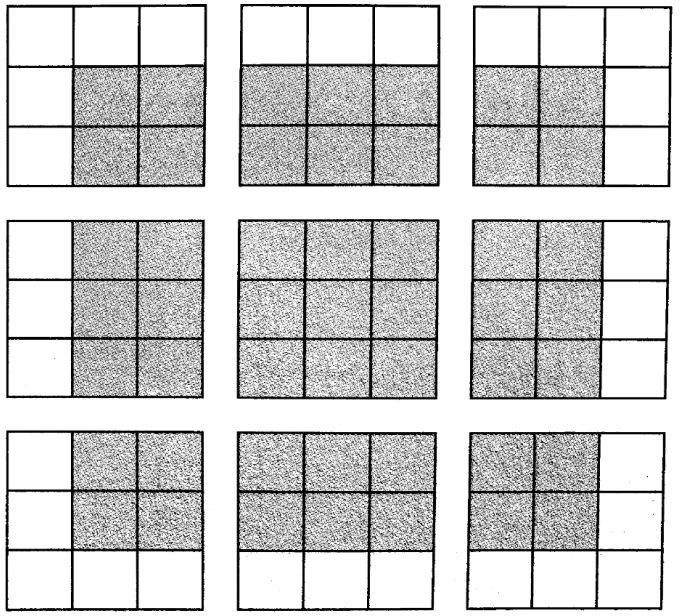
最初畳み込み層のサイズ

入力(**28**, **28**, 1) → 出力(**26**, **26**, 32)

- 5×5の特徴マップの場合

3×3のグリッドを形成するために3×3のウィンドウを中心として使用できるタイルは、9つのみ





したがって、出力特徴マップは3×3となる。

5.1.1 畳み込み演算

畳み込みを定義するパラメータ

- 入力から抽出されたパッチサイズ
  通常3×3か5×5。今回の例では3×3。

- 出力特徴マップの深さ
  計算されるフィルタの数。この例では深さ32で始まり、64で終わる。

KerasのConv2D層では、これらのパラメータは畳み込み層の第一引数となる。

```
Conv2D(output_depth, (window_height, window_width))
```

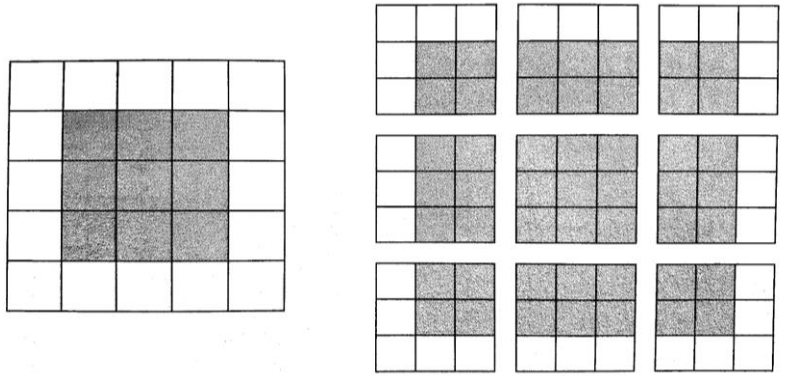
パディング

- パディング

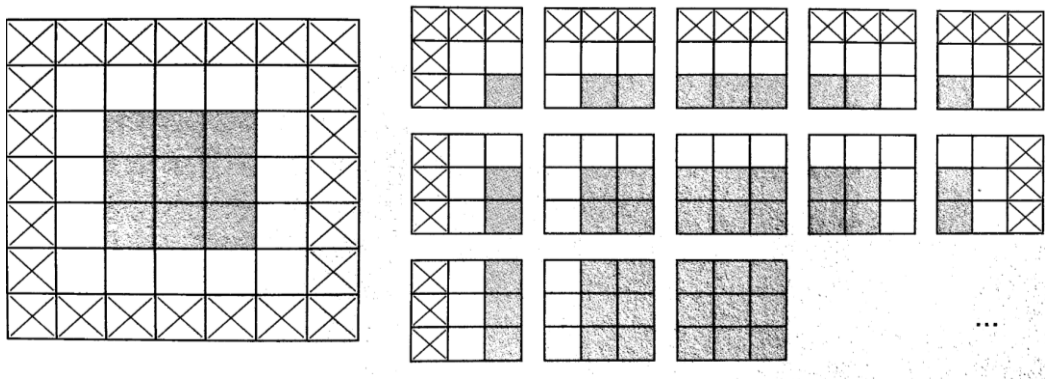
入力特徴マップに行と列を追加することで、畳み込みウィンドウをすべての入力タイルに移動できるようにする、というもの。

入力と同じ空間次元を持つ出力を得られる。(例：(5×5)→(5×5))

- パディングなし

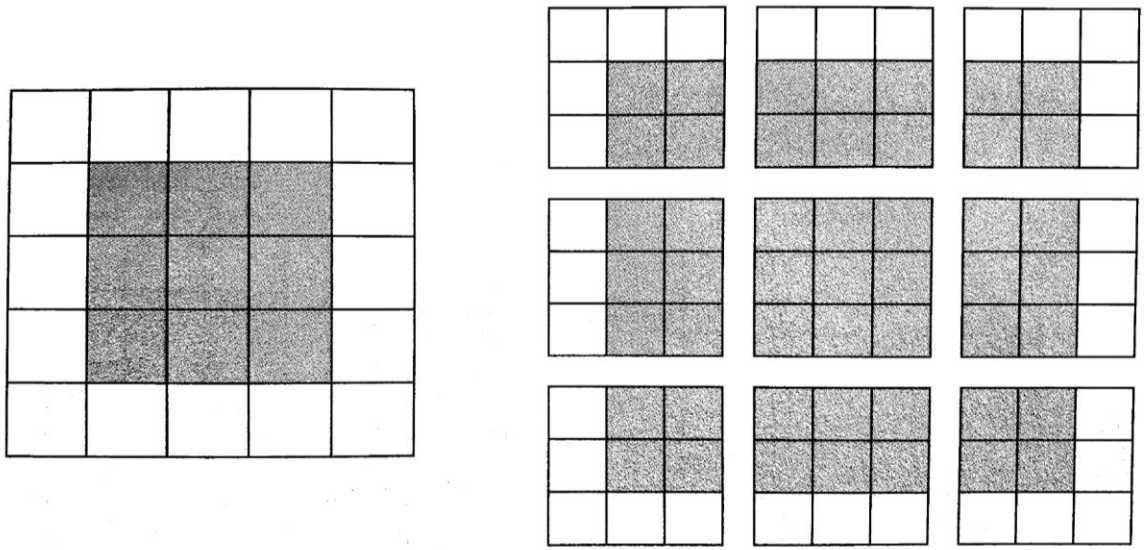


- パディングあり

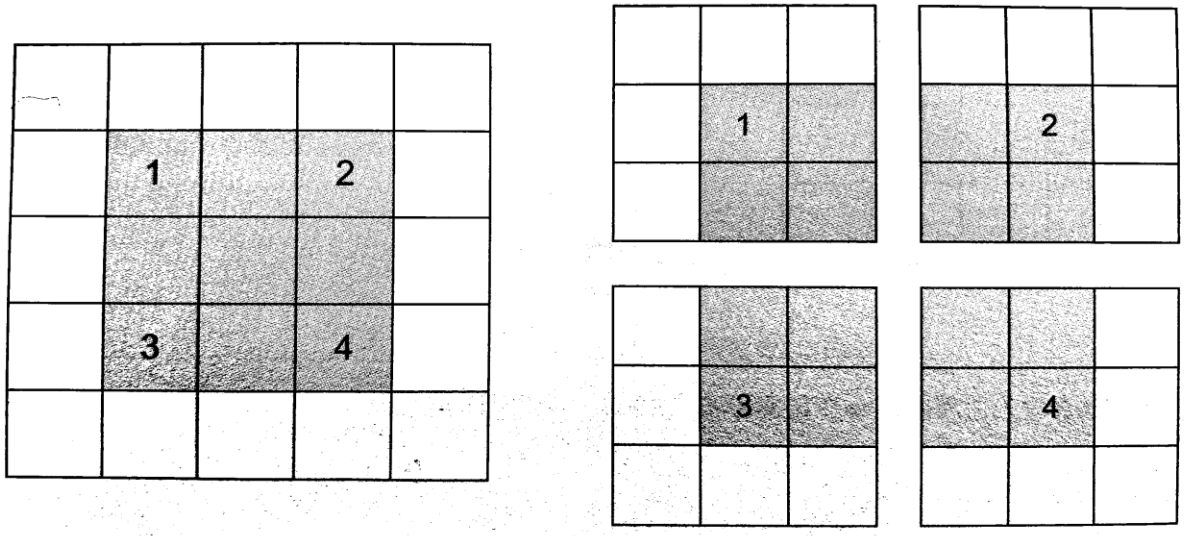


畳み込みのストライド

- ストライド1



- ストライド2



5.1.2 最大値プーリング演算

- 最大値プーリング演算

Layer(type)	**Output shape**	**Param #**
Conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
Maxpooling2d_1 (MaxPooling2D)	(None, **13**, **13**, 32)	0
Conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
Maxpooling2d_2 (MaxPooling2D)	(None, **5**, **5**, 64)	0
Conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928

- 渡される前の特徴マップが、半分に縮小されている。
→ダウンサンプリング

5.1.2 最大値プーリング演算

- ダウンサンプリングする理由は？

大きいマップを維持した場合を考える。→MaxPooling2Dなし

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers. Conv2D(64, (3, 3), activation='relu'))
model.add(layers. Conv2D(64, (3, 3), activation='relu'))
```

5.1.2 最大値プーリング演算

Layer(type)	Output shape	Param #
Conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
Conv2d_5 (Conv2D)	(None, 24, 24, 64)	18496
Conv2d_6 (Conv2D)	(None, 22, 22, 64)	36928

問題点

- 特徴の空間階層の学習に貢献しない
- 最終的な特徴マップが非常に大きい(22×22×64=30,976)

5.1.2 最大値プーリング演算

最大値をとる理由は？

最大値プーリングの代わりに平均値プーリングを使用することもできる。

しかし、最大値プーリングの方がうまくいく傾向にある。

→特徴マップの何らかのパターンの空間的な有無をエンコードする傾向にあるので、平均よりも最大を調べる方が情報利益がある。

最も合理的なサブサンプリング戦略

ストライドなしの畳み込みを通じて特徴量の密なマップを生成。

そして、小さなパッチに対して特徴量の最大活性化を調べる。

入力の疎なウィンドウを調べたり、入力パッチの平均を求めたりすると、特徴量の有無に関する情報を見逃すおそれがある。