

深層学習（3）畳み込みニューラルネットワーク

（高校生・大学初年次向け）

空間的な構造をもつデータののためのネットワーク

Planned and edited by Yoshio Okamoto / Written by Claude Opus 5.5 (Anthropic)

Ver. 0.2 October 5, 2026

※ 以下の内容はAIが生成したものであり、一応編者がチェックしていますが、間違いや思い込みが含まれている場合があることはご注意ください。

© 2026 Yoshio Okamoto. 教育目的での利用は自由です（出典を明記してください）。

はじめに

第1回・第2回で扱ったネットワークは、入力すべての成分を次の層のすべてのユニットにつなぐ全結合のネットワークでした。第3回では、画像のように空間的な並び方に意味があるデータに適した畳み込みニューラルネットワーク（Convolutional Neural Network, CNN）を扱います。

私たちが写真を見るとき、まず目に入るのは「ここに線がある」「ここで色が変わる」といった近くの部分の様子で、それを組み合わせて「これは猫の顔だ」とわかります。CNNは、この「近くを見て、組み合わせる」というやり方を、ネットワークの形として組み込んだものです。

回	題目
第1回	ニューラルネットワークの基礎
第2回	誤差逆伝播法と学習の実際
第3回	畳み込みニューラルネットワーク
第4回	系列データと Transformer

読み方 本文は、高校の数学（数列の和 $\sum$ ，簡単な数の計算）を使って読めるように書きました。畳み込みの数学的な性質（相互相関との違い，平行移動に対する性質の証明，逆伝播の式）や，チャンネルが複数ある場合の一般的な式は，脚注と巻末の付録Aにまとめました。付録は読み飛ばしても本文の流れは追えます。例題の数値はすべて実際に計算した値です（付録Bのプログラム）。

記号の約束

- 1次元のデータ（信号）を $x_0, x_1, \dots, x_{n-1}$ ，2次元のデータ（画像）を X_{ij} （ i 行 j 列の画素）と書きます。添字は0から数えます。
- フィルタ（重み）を w_u （1次元）， W_{uv} （2次元）と書き，その大きさを k （2次元では $k \times k$ ）とします。
- 画像の大きさは「高さ $\times$ 幅 $\times$ チャンネル数」で書きます。チャンネルとは，カラー画像の赤・緑・青のように，同じ位置に重なった複数の値のことです。

1 全結合ネットワークの問題点

1.1 パラメータが多すぎる

224 × 224 画素のカラー画像は、赤・緑・青の3チャンネルがあるので $224 \times 224 \times 3 = 150,528$ 個の数値です。これを全結合の隠れ層（1000ユニット）につなぐと、重みだけで

$$150,528 \times 1000 \approx 1.5 \times 10^8$$

個になります。最初の1層だけで1.5億個のパラメータがあり、学習には膨大なデータが必要で、過学習もしやすくなります。

全結合は、1000人の調査員の一人ひとりが、15万個の画素すべてを見て判断するようなものです。しかも調査員ごとに見方（重み）がばらばらなので、覚えることが膨大になります。

1.2 画像の性質を利用していない

全結合のネットワークは、入力画素を1列に並べて扱うので、どの画素とどの画素が隣り合っているかという情報を使っていません。画素の順番を入れ替えても、ネットワークにとっては同じ問題です。写真をジグソーパズルのようにばらばらにして並べ替えても、全結合のネットワークは気にしないのです。

しかし画像には次のような性質があります。

1. **局所性**：意味のある特徴（輪郭、角、模様）は、近くの画素どうしの関係で決まる。
2. **位置によらない**：猫の目は、画像の左上にあっても右下にあっても猫の目である。

この2つの性質を構造として組み込んだのが、畳み込みニューラルネットワークです。

2 1次元の畳み込み

2.1 定義

まず1次元で考えます。長さ n の信号 x_i と、長さ k のフィルタ w_u の畳み込み（convolution）を

$$y_i = \sum_{u=0}^{k-1} w_u x_{i+u} + b, \quad i = 0, 1, \dots, n-k$$

と定義します¹。たとえば $k = 3$ なら

$$y_0 = w_0x_0 + w_1x_1 + w_2x_2 + b, \quad y_1 = w_0x_1 + w_1x_2 + w_2x_3 + b, \quad \dots$$

です。

フィルタを信号の上で1つずつずらしながら、重なった部分の積の和をとる操作です。出力の長さは $n - k + 1$ です。

¹数学や信号処理でいう「畳み込み」は、フィルタを左右反転させてから重ねる演算で、ここでの定義は正確には「相互相関」と呼ばれます。深層学習ではどちらも違いがないので、反転しない形を慣習として「畳み込み」と呼んでいます（付録A.1）。

細長いのぞき窓（ものさし）を信号の上で端から端まで滑らせ、窓から見える数か所の値に決まった重みを掛けて足す——それを位置ごとに記録していくイメージです。フィルタは、「どんな形を探しているか」を表す型紙だと考えてもかまいません。

全結合と比べると、畳み込みは次の2つの制約をもった特別な全結合層と見るができます（図1）。

- ・ **局所結合**：各出力は、入力近くの k 個だけにつながる。
- ・ **重み共有**：どの位置の出力も、同じ k 個の重み $w_0, \dots, w_{k-1}$ を使う。

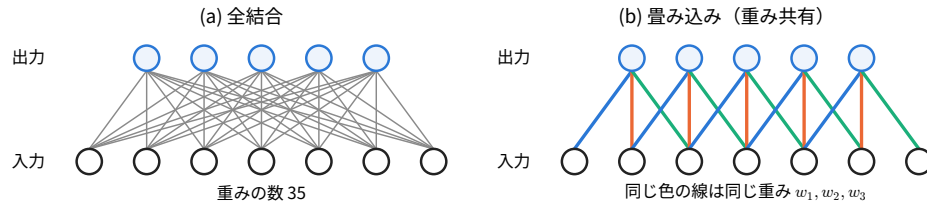


図1 (a) 全結合と (b) 畳み込みの比較（入力7，出力5，フィルタの長さ3）。全結合では35個の重みがすべて別々だが，畳み込みでは3個の重みを全位置で使い回す。

先ほどの調査員のたとえで言えば，畳み込みでは，各調査員は自分の近所の数画素だけを見て（局所結合），しかも全員が同じマニュアル（同じ重み）で判断します（重み共有）。覚えることがずっと少なくて済むのです。

2.2 例：平滑化と差分

8個の値からなる段差のある信号 $x = (0, 0, 0, 0, 3, 3, 3, 3)$ に，2種類のフィルタをかけます（付録Bのプログラム）。

フィルタ	w	出力 y
移動平均	$(1/3, 1/3, 1/3)$	$(0, 0, 1, 2, 3, 3)$
差分	$(-1, 0, 1)$	$(0, 0, 3, 3, 0, 0)$

たとえば移動平均の y_2 は $\frac{1}{3}(x_2 + x_3 + x_4) = \frac{1}{3}(0 + 0 + 3) = 1$ ，差分の y_2 は $-x_2 + 0 \cdot x_3 + x_4 = -0 + 3 = 3$ です。

- ・ **移動平均**のフィルタは，隣り合う3つの値を平均し，急な変化をなだらかにします。気温の毎日の値から「7日間の移動平均」をとって細かい上下をならすのと同じです。
- ・ **差分**のフィルタは $y_i = x_{i+2} - x_i$ を計算し，**値が変化している場所**でだけ大きな値を出します。段差（エッジ）の検出器です。

雑音のある信号では，図2のようになります。差分フィルタを直接かけると雑音も強調されますが，移動平均で雑音をならしてから差分をとると，段差の位置がはっきり現れます。

このような操作は，信号処理では**デジタルフィルタ**（FIR フィルタ）として昔から使われてきました。地震計の記録から雑音を取り除いたり，波の到着の瞬間を探したりするのも，同じ種類の計算です。CNNの新しさは，**フィルタの係数を人が設計するのではなく，データから学習する点**にあります。

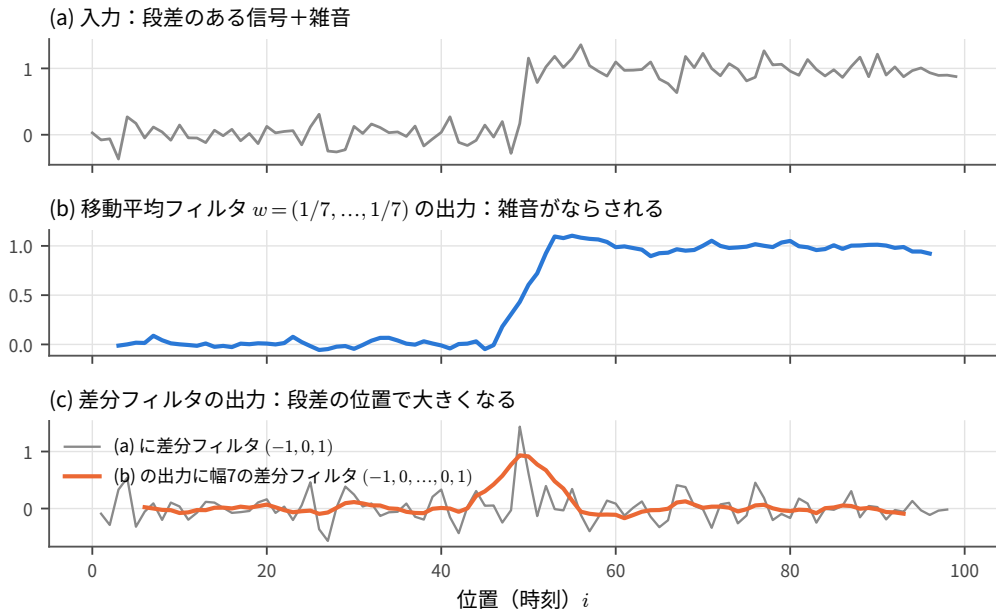


図2 1次元の畳み込みの例. (a) 位置50に段差のある信号に雑音を加えたもの. (b) 7点の移動平均. (c) 差分フィルタの出力 (灰色：(a) に直接かけたもの、橙：(b) に幅7の差分フィルタをかけたもの) .

3 2次元の畳み込み

3.1 定義

画像 X と $k \times k$ のフィルタ W の畳み込みは

$$Y_{ij} = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} W_{uv} X_{i+u, j+v} + b$$

です. $\sum$ が2つあるのは、フィルタの縦方向 (u) と横方向 (v) の両方について足すため、 3×3 のフィルタなら9個の積の和になります.

$k \times k$ の窓を画像の上で縦横にずらしながら、重なった部分の積の和をとります (図3) . 虫眼鏡で写真を左上から右下まで少しずつ見て回り、それぞれの場所で「探している形にどれだけ似ているか」を点数にして書き留めていく作業です.

図3のフィルタ

$$W = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

は、「右の列から左の列を引く」フィルタで、**縦方向の境界**を検出します. 出力は、入力の0と1の境界にあたる列だけが3になり、一様な部分では0になります. このフィルタの縦と横を入れ替えたもの (「下の行から上の行を引く」フィルタ)²は**横方向の境界**を検出し、図3の入力 (縦の境界しかない) に対する出力はすべて0になります.

図4は、もう少し複雑な画像にこれらのフィルタをかけた例です.

境界の検出は、たとえば衛星画像から海岸線や雲の縁を取り出すときにも使われる、画像処理の基本的な操作です.

²行列の行と列を入れ替える操作を**転置**といい、 W^T と書きます.

3.2 出力の大きさ：パディングとストライド

- ・ **パディング (padding)** : 入力周囲を幅 p だけ0で埋めてから畳み込みます. 写真に額縁をつけるようなもので, 窓が画像の端からはみ出しても計算できるようになります. $k=3$ なら $p=1$ とすると, 出力が入力と同じ大きさになります.
- ・ **ストライド (stride)** : 窓を1画素ずつではなく s 画素ずつずらします. 飛び石を1つおきに渡るように, $s=2$ なら出力の縦横はおよそ半分になります.

入力の大きさ (縦または横の画素数) が n のとき, 出力の大きさは

$$\frac{n + 2p - k}{s} \text{ の小数点以下を切り捨てた数 } + 1$$

です. パディングで幅が $n + 2p$ になり, そこに大きさ k の窓を置ける場所の数を数えています. 図3では $n=6$, $k=3$, $p=0$, $s=1$ なので $6 - 3 + 1 = 4$ です.

3.3 複数のチャネル

実際のCNNでは, 入力も出力も複数のチャネルをもちます.

- ・ 入力が C_{in} チャネルのとき, 1つのフィルタは $k \times k \times C_{\text{in}}$ の大きさを持ち, 全チャネルにわたって積の和をとります (カラー画像なら, 赤・緑・青それぞれの $k \times k$ の窓の積の和を, さらに3チャネル分足し合わせます).
- ・ このフィルタを C_{out} 個用意すれば, 出力は C_{out} チャネルになります.

チャネルは, 同じ地域の地図を何枚も重ねたものと考えerとわかりやすいでしょう. 天気図に気温・気圧・湿度の地図があるように, カラー画像には赤・緑・青の3枚の「地図」が重なっています. フィルタは, この重なった地図を同時に見て, 1枚の新しい地図 (特徴マップ) を作ります.

式で書くと添字が4つの量が出てきます (付録A.2). パラメータの数は, フィルタ1個あたり $k \times k \times C_{\text{in}}$ 個の重みと1個のバイアスなので,

$$k \times k \times C_{\text{in}} \times C_{\text{out}} + C_{\text{out}}$$

です. たとえばカラー画像 ($C_{\text{in}}=3$) に 3×3 のフィルタを64個かけると, $3 \times 3 \times 3 \times 64 + 64 = 1,792$ 個です.

パラメータの数が画像の大きさによらない点に注目してください. 1.1節の全結合の1.5億個と比べると, 桁違いに少なくなっています.

各出力チャネルは, それぞれ別のフィルタに対応するので, 「縦の境界」「横の境界」「特定の色」など, 異なる種類の特徴を検出した地図 (特徴マップ, feature map) になります.

4 畳み込みの性質

4.1 重み共有と平行移動

同じフィルタを全位置で使うので, 入力をずらすと, 出力も同じだけずれます.

たとえば2.2節の段差のある信号を右に1つずらして $(0, 0, 0, 0, 0, 3, 3, 3)$ にすると, 差分フィルタの出力も $(0, 0, 0, 3, 3, 0)$ と, 右に1つずれます (一般的な証明は付録A.3). この性質を平

行移動に対する同変性 (translation equivariance) といいます。画像のどこに猫の目があっても、同じフィルタがそこで反応するのです。これが1.2節の性質2を構造として組み込んだものです。

金属探知機を持って砂浜を歩くと、探知機はどこでも同じ感度で反応します。コインが右に1m ずれて埋まっていれば、反応する場所も右に1m ずれるだけです。同じフィルタ (同じ探知機) を全位置で使うことの効果がこれです。

4.2 受容野

3×3 の畳み込みを1回かけた出力の1画素は、入力 3×3 の範囲の情報だけを使います。この範囲を受容野 (receptive field) といいます。 3×3 の畳み込みを2回重ねると受容野は 5×5 、 L 回重ねると $(2L + 1) \times (2L + 1)$ に広がります (1回重ねるごとに、上下左右に1画素ずつ広がるからです)。

層を重ねるほど、広い範囲の情報を組み合わせた特徴を表せるようになります。町内会の班長は自分の班の数軒を見ていれば十分ですが、その班長たちの報告をまとめる町内会長は町内全体を、さらに町内会長たちの報告をまとめる人は市全体を見渡せます。一人ひとりが見る範囲は狭くても、報告を積み重ねることで、上の層ほど広い範囲がわかるのです。

5 プーリング

プーリング (pooling) は、特徴マップを小さな領域に区切り、各領域を1つの値で代表させて縮小する操作です。最もよく使われるのは、各領域の最大値をとる最大値プーリングです (図5)。各班から、いちばん強く反応した値を代表として選ぶようなものです。

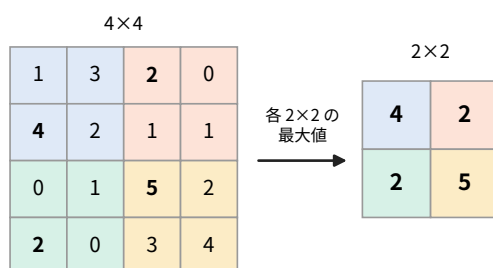


図5 2×2 の最大値プーリング (ストライド2)。 4×4 の特徴マップが 2×2 になる。

プーリングには次の効果があります。

- ・ 特徴マップが縦横それぞれ半分になり、後の層の計算量が減る。
- ・ 受容野が速く広がる。
- ・ 特徴の位置が1画素程度ずれても出力が変わりにくくなる (小さな位置ずれに対する頑健さ)。

プーリングには学習するパラメータがありません。

地図の縮尺を小さくするのに似ています。縮小した地図では細かい道の位置はわからなくなりますが、「このあたりに駅がある」ことはわかります。「何があるか」は残しつつ、「正確にどこか」の細かい情報を捨てて、全体を見やすくするのです。

6 CNN の全体構成

6.1 典型的な構成

典型的な CNN は、「畳み込み → ReLU → プーリング」を何段か繰り返して特徴を取り出し、最後に全結合層とソフトマックス関数で分類します（図6）。

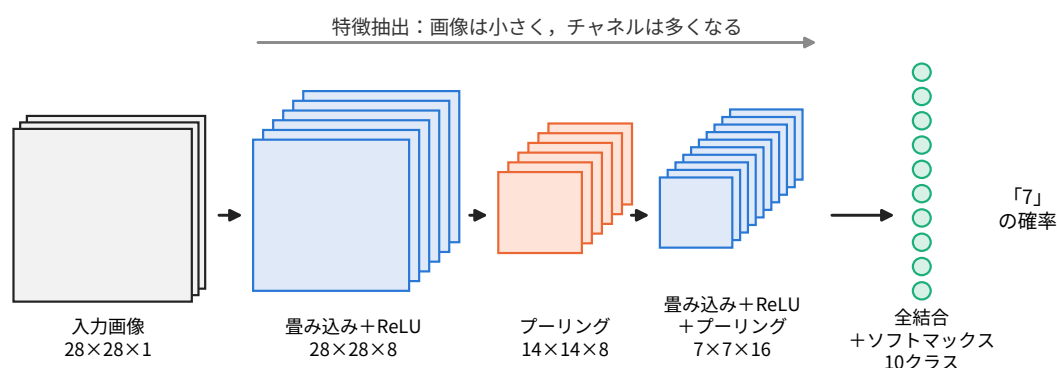


図6 手書き数字（ 28×28 画素）を分類する小さな CNN の構成例。畳み込み層は 3×3 のフィルタ（パディング1）、プーリングは 2×2 の最大値プーリング。

図6のネットワークのパラメータ数は次のとおりです。

層	計算	パラメータ数
畳み込み1（ $1 \rightarrow 8$ チャンネル）	$3 \times 3 \times 1 \times 8 + 8$	80
畳み込み2（ $8 \rightarrow 16$ チャンネル）	$3 \times 3 \times 8 \times 16 + 16$	1,168
全結合（ $7 \times 7 \times 16 = 784 \rightarrow 10$ ）	$784 \times 10 + 10$	7,850
合計		9,098

第1回全結合ネットワーク（ $784 \rightarrow 100 \rightarrow 10$ ）の79,510個と比べると、約9分の1です。

パラメータ（つまみ）が少なく、画像の性質に合った構造をもつので、少ないデータでも高い精度が得られます。

6.2 学習

CNN も第2回の誤差逆伝播法で学習します。第2回で見たように、重みの勾配は「後ろから届いた誤差 $\times$ その重みに入ってきた値」でした。畳み込み層では同じ重みが全位置で使われているので、**重みの勾配は、各位置でのこの値をすべて足し合わせたもの**になります（式は付録 A.4）。同じマニュアルで働く調査員全員の「直してほしい点」を集計して、マニュアルを1

回改訂する，という感じです。最大値プーリングでは，逆伝播の誤差は各領域で最大値をとった位置（代表に選ばれた値）にだけ送られます。

6.3 何を学習しているか

学習済みの CNN を調べると，最初の層のフィルタは図4のような**境界や色の変化の検出器**に似たものになることが多く，深い層ほど**模様，部品（目，車輪など），物体全体**といった，より複雑で抽象的な特徴に反応するようになることが観察されています。

第1回で見た「隠れ層が表現を変換する」ことが，画像では**単純な特徴から複雑な特徴への階層**として現れるのです。線が集まって角や曲線になり，角や曲線が集まって目や耳になり，目や耳が集まって顔になる——人が境界検出フィルタを設計しなくても，分類という目的のために必要なフィルタが学習によって得られます。

6.4 歴史と応用

- ・手書き数字の認識に CNN を使った LeNet（LeCun ら，1998年）が初期の代表例です。
- ・2012年に，大規模な CNN（AlexNet）が GPU を使って学習され，約120万枚の画像を1000種類に分類する競技で従来の手法を大きく上回ったことが，現在の深層学習の流行のきっかけになりました。
- ・画像の分類のほか，物体の位置の検出，医用画像の診断支援，衛星画像の解析などに使われています。
- ・1次元の CNN は，音声や各種の時系列データ（たとえば地震波形からの波の到着の検出）にも使われています。

7 まとめ

1. 全結合ネットワークは，画像のような大きな入力ではパラメータが膨大になり，画素の空間的な並びも利用しない。
2. 畳み込みは「局所結合」と「重み共有」をもった特別な層で，同じフィルタを全位置で使い回す。パラメータ数は入力の大きさによらない。
3. フィルタは平滑化や境界検出などの特徴抽出器として働き，入力をずらすと出力も同じだけずれる（平行移動に対する同変性）。
4. プーリングは特徴マップを縮小し，小さな位置ずれに対する頑健さを与える。
5. CNN は「畳み込み → ReLU → プーリング」を重ねて，単純な特徴から複雑な特徴を階層的に学習する。フィルタは人が設計せず，誤差逆伝播法で学習する。

次回予告 画像は空間的な並びをもつデータでしたが，文章や時系列は**時間的な順序**をもつデータです。第4回では，系列データを扱う RNN と，現在の LLM の基盤である Transformer を扱います。

付録A 畳み込みの数学

この付録では，偏微分と多くの添字をもつ量を使います。

A.1 「畳み込み」と「相互相関」

数学や信号処理でいう畳み込みは

$$(w * x)_i = \sum_u w_u x_{i-u}$$

と、フィルタを**反転**させて重ねます（* は畳み込みの演算記号で、掛け算ではありません）。本文の定義 $\sum_u w_u x_{i+u}$ は、正確には**相互相関**（cross-correlation）と呼ばれる演算です。深層学習ではフィルタ w を学習で求めるので、反転させてもさせなくても、学習結果の w が反転するだけで違いはありません。そのため、慣習として反転しない形を「畳み込み」と呼んでいます。

A.2 複数チャネルの畳み込み

入力 C_{in} チャネル、出力 C_{out} チャネルのとき、出力チャネル c' の (i, j) 成分は

$$Y_{c', i, j} = \sum_{c=1}^{C_{\text{in}}} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} W_{c', c, u, v} X_{c, i+u, j+v} + b_{c'}$$

です。 W は4つの添字をもつ**4階テンソル**で、その成分の数 $k \times k \times C_{\text{in}} \times C_{\text{out}}$ が重みの数です。

出力の大きさ 本文3.2節の出力の大きさは、床関数（ $[a]$ は a を超えない最大の整数）を使って $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1$ と書きます。

A.3 平行移動に対する同変性

入力を m だけ右にずらしたものの $x'_i = x_{i-m}$ に対する出力は

$$y'_i = \sum_u w_u x'_{i+u} = \sum_u w_u x_{i-m+u} = y_{i-m}$$

となり、元の出力を m だけずらしたものになります（バイアスは省略しました）。

A.4 畳み込み層の逆伝播

1次元で、出力 $y_i = \sum_u w_u x_{i+u} + b$ の重み w_u は、すべての位置 i の出力に使われています。連鎖律の「すべての経路について足す」により、

$$\frac{\partial L}{\partial w_u} = \sum_i \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_u} = \sum_i \frac{\partial L}{\partial y_i} x_{i+u}$$

です。 $\partial L / \partial y_i$ が「位置 i に後ろから届いた誤差」、 x_{i+u} が「その位置で重み w_u に入ってきた値」で、第2回の「誤差 $\times$ 入ってきた値」を全位置について足したものになっています。

付録B Python コード（標準ライブラリのみ）

本文の2.2節，3.1節，5節の例を計算するプログラムです．外部ライブラリは使わず，Python 3 の標準機能だけで書いてあります．

```
# 畳み込みとプーリング（標準ライブラリのみ）

def conv1d(x, w):
    """1次元の畳み込み（パディングなし，ストライド1）"""
    k = len(w)
    return [sum(w[u] * x[i + u] for u in range(k)) for i in range(len(x) - k + 1)]

def conv2d(img, w):
    """2次元の畳み込み（パディングなし，ストライド1）"""
    H, W, k = len(img), len(img[0]), len(w)
    return [[sum(w[u][v] * img[i + u][j + v] for u in range(k) for v in range(k))
              for j in range(W - k + 1)] for i in range(H - k + 1)]

def maxpool(a, s=2):
    """s×s の最大値プーリング（ストライド s）"""
    return [[max(a[i + u][j + v] for u in range(s) for v in range(s))
              for j in range(0, len(a[0]), s)] for i in range(0, len(a), s)]

def show(name, a):
    print(name)
    for row in a:
        print(" " + " ".join(f"{v:3g}" for v in row))

# --- 1次元：段差のある信号 ---
x = [0, 0, 0, 0, 3, 3, 3, 3]
print("入力      ", x)
print("移動平均  ", [round(v, 2) for v in conv1d(x, [1/3, 1/3, 1/3])])
print("差分      ", conv1d(x, [-1, 0, 1]))

# --- 2次元：縦の境界がある 6×6 の画像 ---
img = [[0, 0, 0, 1, 1, 1] for _ in range(6)]
vert = [[-1, 0, 1],
         [-1, 0, 1],
         [-1, 0, 1]] # 縦の境界を検出するフィルタ
horiz = [list(r) for r in zip(*vert)] # 縦横を入れ替えると横の境界用
show("入力画像", img)
show("縦境界フィルタの出力", conv2d(img, vert))
show("横境界フィルタの出力", conv2d(img, horiz))

# --- 最大値プーリング ---
a = [[1, 3, 2, 0],
      [4, 2, 1, 1],
      [0, 1, 5, 2],
      [2, 0, 3, 4]]
show("プーリング前", a)
show("2×2 最大値プーリング後", maxpool(a))
```

実行結果：

```
入力      [0, 0, 0, 0, 3, 3, 3, 3]
移動平均  [0.0, 0.0, 1.0, 2.0, 3.0, 3.0]
```

差分 [0, 0, 3, 3, 0, 0]

入力画像

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

縦境界フィルタの出力

0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0

横境界フィルタの出力

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

プーリング前

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

2×2 最大値プーリング後

4	2
2	5