# 第 1 章

# 畳み込みニューラルネットワーク

**畳み込み**とよばれる処理を用いたニューラルネットワークを**畳み込みニューラルネットワーク**（convolutional neural network, CNN）という。畳み込みニューラルネットワークは脳の視覚野の研究から生まれたもので、現在では画像認識や音声認識、自然言語処理に用いられる。ここでは畳み込み処理といくつかの応用を説明する。

## 1.1 畳み込み処理

実数を引数にとる実関数 $I(t)$ と重みの関数 $K(t)$ を用いて

$$S(t) = \int I(\tau)K(t-\tau)\,\mathrm{d}\tau$$

という関数 $S(t)$ を生成する。この処理を**畳み込み**（convolution）とよび、$S(t) = (I * K)(t)$ のように表記する。これらの関数の引数が離散値であるならば、離散畳み込みを

$$S(t) = (I * K)(t) = \sum_{\tau} I(\tau)K(t-\tau)$$

と定義する。畳み込みネットワークの文脈では $I$ を**入力**（input）、$K$ を**カーネル**（kernel）または**フィルター**（filter）、$S$ を**特徴マップ**（feature map）とよばれる。

2 変数に対する離散畳み込みは

$$S(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n)$$

となる。ただし畳み込みの可換性 $(I * K)(i,j) = (K * I)(i,j)$ を用いたため、フィルター $K(m,n)$ ではなく入力 $I(i+m, j+n)$ についての和をとっている。実装上ではこのように計算する方が容易である。一方で、ニューラルネットワークの実装においては**相互相関**（cross-correlation）とよばれる次の量も畳み込みとよび、よく用いられている。

$$S(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m,n).$$

2 次元のデータに畳み込み処理を行う例を図 1.1 に示す。簡単のため入力 $I$ はサイズが $6 \times 6$ の行列であるとし、その成分の値をグレースケールで視覚化してある。2 次元のフィルター $K$ は $3 \times 3$ の行列で

ある。フィルターはその中心を決められるように、行数と列数ともに奇数であると便利である。まず図 1.1 中の緑色の四角形で示したように、入力 $I$ の左上の位置にフィルター $K$ の成分が全て含まれるように重ねる。同じ位置にある成分同士を掛け合わせ、総和をとると 4 になるので、その値が特徴マップ $S$ の左上の成分になる。次に図 1.1 中の赤色の四角形で示したように、重ね合わせていたフィルター $K$ を右に 1 つずらして同様の計算をする。入力 $I$ の右端までこの操作を繰り返したら 1 行下がる。この操作を入力 $I$ の右下に達するまで繰り返す。なお、入力 $I$ に重ねているフィルター $K$ の影のようなこの領域を、神経科学の言葉になぞらえて、**受容野**（receptive field）とよぶ。

より一般に、2 次元の入力 $I$ のサイズが $N \times M$, 2 次元のフィルター $K$ のサイズが $(2k+l) \times (2l+1)$ であるとする。ただし $N$, $M$, $k$, $l$ は 0 を含まない自然数である。$(i, j)$ 成分の値をそれぞれ $I(i, j)$, $K(i, j)$ としたとき、2 次元畳み込み処理によって得られる値 $S(i, j)$ は

$$S(i,j) = \sum_{n=0}^{2l} \sum_{m=0}^{2k} I(i+m, j+n) K(1+m, 1+n)$$

となる。$i$ と $j$ の範囲は $i = 1, \ldots, M-2k$, $j = 1, \ldots, N-2l$ で、特徴マップ $S$ のサイズは $(M-2k) \times (N-2l)$ となる。ただしコンピュータ言語 Python では配列の要素番号は 0 から始まるので、その形式に合わせるには総和の下端を $n = 0$ から $n = -1$ のように書き換えればよい。

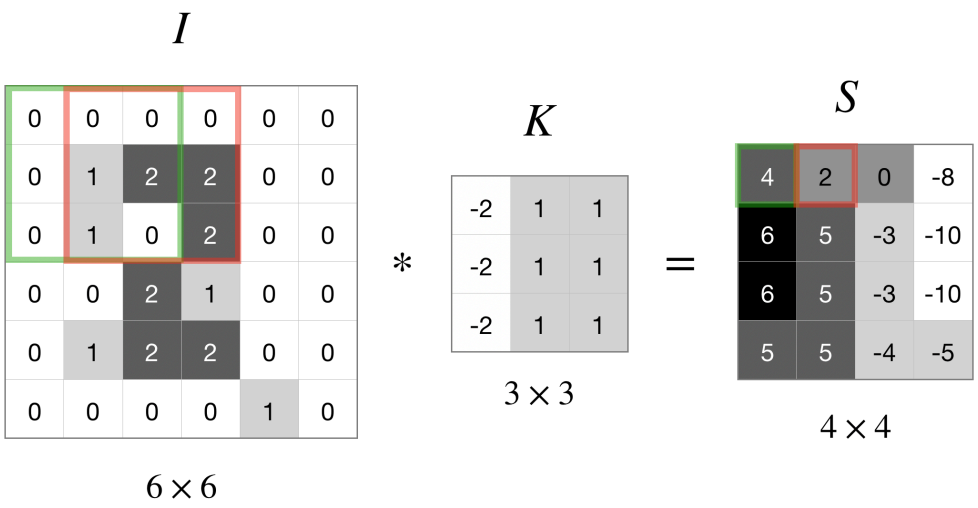


図 1.1　2 次元畳み込み処理の例

## 1.2　ゼロパディングとストライド

図 1.1 の例では、入力のサイズが $6 \times 6$ であったのに対し、出力のサイズは $4 \times 4$ に縮小している。これでは畳み込みを行うごとに出力が縮小していってしまう。このことを防ぐには、図 1.2 のようにあらかじめ入力の周囲にゼロを追加すればよい。この「余白」のように追加された部分を**ゼロパディング**（zero padding）とよぶ。図 1.2 の例ではゼロパディングの幅は 1 にしているが、この幅はフィルター $K$ のサイズによって変化させればよい。そのサイズが $(2k+l) \times (2l+1)$ ならば、入力 $I$ の左右に幅 $k$, 上下に幅 $l$ のゼロパディングをそれぞれ追加する。

また受容野をずらす間隔のことを**ストライド**（stride）という。図 1.1 と図 1.2 の例ではストライドは 1 であった。ストライドを大きくすると出力は小さくなる。

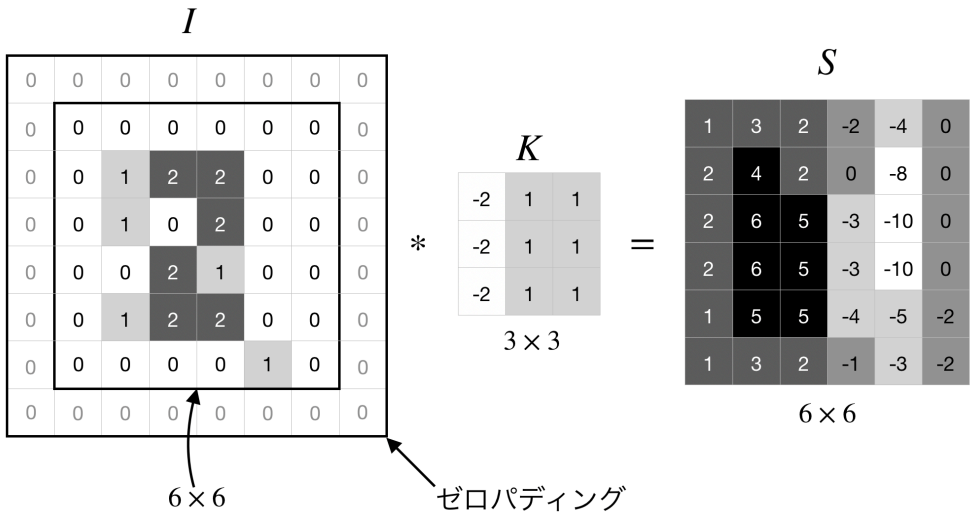


図 1.2　2 次元畳み込み処理の例

ゼロパディングの幅やストライドの値は、ライブラリで畳み込みネットワークを使用する際、引数として渡す。

## 1.3　フィルターの効果

フィルターを使うと、入力のフィルターに近い部分が強調されて特徴マップとして得られる。

図 1.1 と図 1.2 の例におけるフィルター

$$K = \begin{bmatrix} -2 & 1 & 1 \\ -2 & 1 & 1 \\ -2 & 1 & 1 \end{bmatrix} \tag{1.1}$$

について考える。入力 $I$ において、左側で小さい値を持ち、右側で大きい値を持つような領域をこの受容野が通ったとき、特徴量マップの対応するピクセルは大きな値を持つが、反対に左側で値が大きく右側で小さい領域を受容野が通ったときは、特徴量の値は小さくなる。左右の方向に均一な入力の領域では、このフィルターを使うと 0 に近い値を返す。このように、式 (1.1) のフィルターは入力の縦線の右側のエッジを強調させる効果をもつ。なおこのフィルターは全要素の総和が 0 になるように設計した。これにより、強調させたい入力の特徴を 0 以上として定めることができる。

同様に様々なフィルターを考案することができる。例えば

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

はそれぞれ横線、内側の点、右肩上がりの斜め線を強調するようなフィルターとなる。

畳み込みニューラルネットワークでは、フィルターを重みとして訓練する。

このように、フィルターは入力のある特徴を誇張させる効果がある。畳み込みニューラルネットワークは訓練によってタスクに役立つフィルターを発見し、それらの組み合わせを学習していく。

## 1.4　プーリング

フィードフォーワード・ニューラルネットワークで手書き数字を認識する場合、2 次元の画像データをベクトルに変換してから学習する。したがってそのモデルは入力が 2 次元の画像であったことを知らず、データの変換の仕方には自由度があるため、画素の並ぶ順番を変えても全く同じ精度で学習できてしまう。

畳み込み処理は、データが 2 次元であるという特徴を活用できている。もう一つの活用方法として、**プーリング**（pooling）という方法がある。プーリングを使えば入力画像を**サブサンプリング**（subsampling, 縮小）して計算の負荷、メモリ使用量、パラメータ数を削減し、過学習を抑制することができる。

プーリングとは 2 次元の画像のある小領域に着目し、その領域内の最大値や平均値など、小領域に対応した値を返し新たな 2 次元のデータを作り出すような処理である。プーリングの例を図 1.3 に示す。入力は $6 \times 6$ の画像で、**プーリングカーネル**（pooling kernel）のサイズは $2 \times 2$，出力のサイズは $3 \times 3$ になる。$2 \times 2$ のプーリングカーネルに着目し、その領域の中で最大の数値を出力値としている。この例ではストライドを 2 として同様の処理を繰り返す。このようにプーリングカーネルの最大値を出力とするプーリングを**最大プーリング**（max pooling）という。この他にもプーリングカーネルの平均値を出力とする方法もあり、それを**平均プーリング**（average pooling）という。

プーリングの利点は少しずれた入力に対しても似たような出力が得られることである。人間の目は少しずれた画像に対しても同じ認識ができるが、プーリングによってそのような性質をネットワークに持たせることができる。

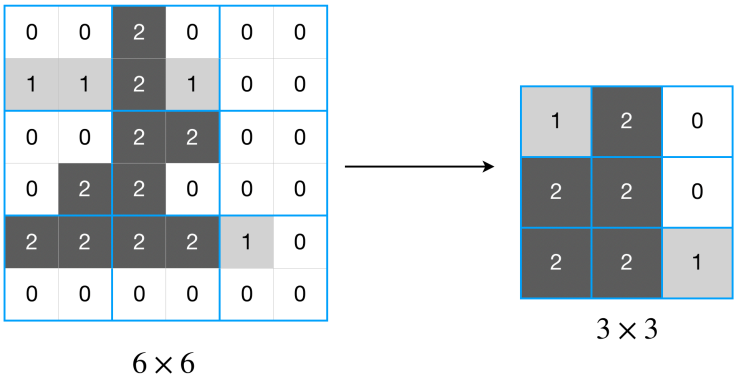


図 1.3　最大プーリングの例

## 1.5　ドロップアウト

よく使われる深層ニューラルネットワークを改善する正則化の方法の 1 つとして、**ドロップアウト**（dropout）がある。各訓練ステップで隠れ層のユニットを確率 $p$ でランダムに選び、それらを「脱落」させて存在しないものとして扱い、学習をさせるのである。この確率 $p$ は人間が調節するハイパーパラメータで、**ドロップアウト率**（dropout rate）とよばれる。訓練後はすべてのニューロンを参加させるが、そうすると出力が $1/p$ 倍に大きくなってしまうので、出力を $p$ 倍するなどして辻褄を合わせることが重要である。

ドロップアウトでニューラルネットワークの性能が上がることは、会社の業績に例えられることがある

([?] など)。社員は毎朝コイントスで出社するかどうかを決め、その日出社した社員たちは欠席している社員たちの穴を埋めるように協力して業績を上げなければいけない。そのような日常に慣れている社員たちにとっては、全員が出社した日の仕事はとても簡単に感じるだろう。実際の会社でこの考え方が通用するかどうかはわからないが、ニューラルネットワークでは機能するのだ。

## 1.6　畳み込みニューラルネットワークのアーキテクチャ

一般的な畳み込みニューラルネットワークのアーキテクチャは、いくつかの畳み込み層の後にプーリング層、さらにいくつかの畳み込み層、プーリング層…と積み上げていく。最後の層の後は通常のフィードフォーワード・ニューラルネットワークが追加され、例えばソフトマックス関数を通して、出力層となる。

図 1.4 は、手書きの数字の認識で広く使われている畳み込みニューラルネットワークのアーキテクチャである LeNet-5 を図示したものである [?]。訓練データのサイズは $28 \times 28$ だが、ゼロパディングを追加して $32 \times 32$ としたものを入力にする。$5 \times 5$ のフィルターを使って $28 \times 28$ の畳み込み層を 6 枚出力する（C1)。次に $2 \times 2$ のプーリングカーネルで $14 \times 14$ のプーリング層を 6 枚出力する（S2)。同様のことを繰り返して畳み込み層を 3 層、プーリング層を 2 層形成している。

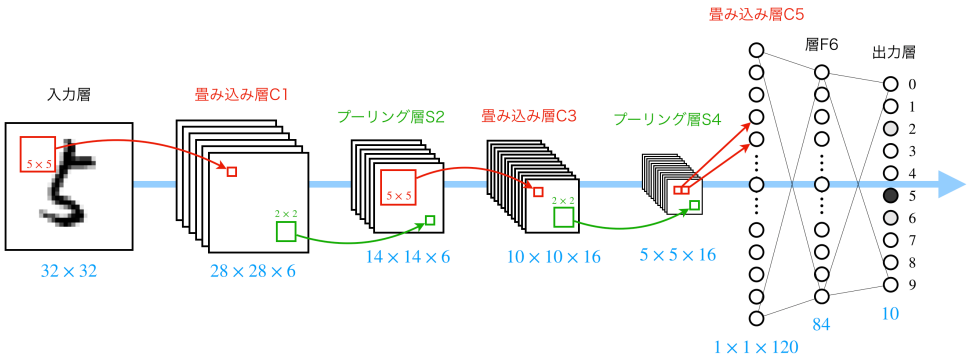


図 1.4　LeNet-5