

立教大学大学院人工知能科学研究科

認識技術特論 畳込みネットワーク

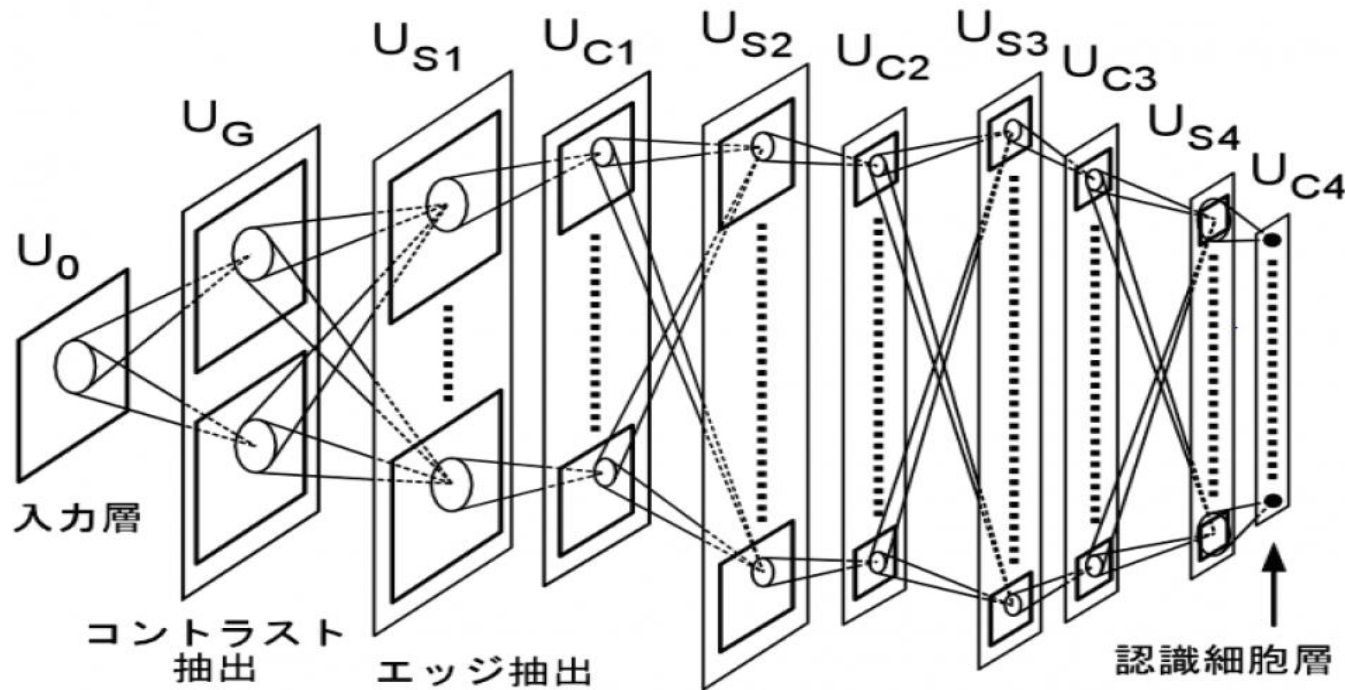
担当： 韓 先花（ハン）

今日の内容

- **深層学習の発展**
- 畳込みネットワークの構造
- 基本的な演算

ネオコグニトロン： Neocognitron

1980年代に福島邦彦によって提唱された階層的、多層化された[人工ニューラルネットワーク](#)である



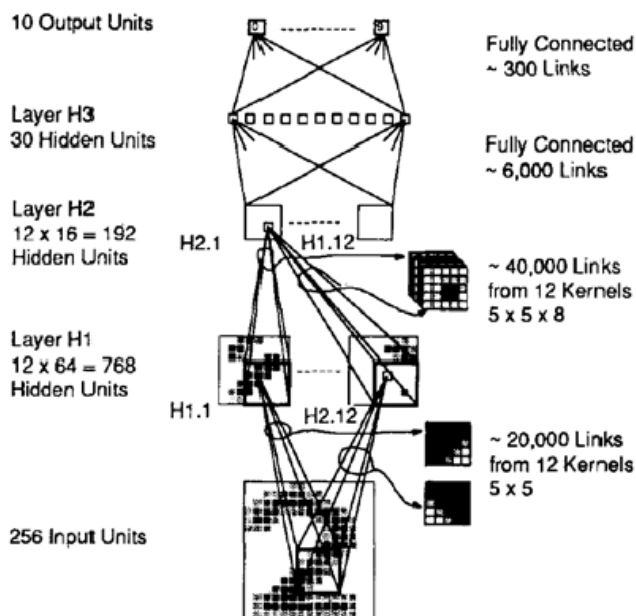
人間の脳を模倣

- ・ 畳み込みネットの初期型
 - ・ 自己組織型学習
- 素子を足してゆく

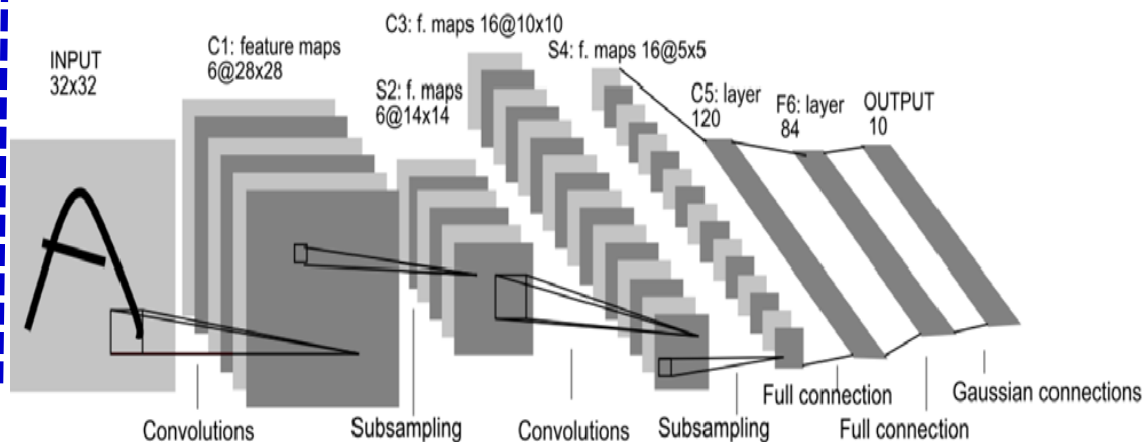
LeNet (LeCun)

LeCun, Hinton, Bengio: AI conspirators
awarded prestigious Turing prize

[LeCun+etal,89]



LeNet-5
[LeCun etal,98]



- 畳み込み + プーリング：現在も使われている構造
- 誤差逆伝搬法でパラメータを更新
- 手書き文字認識データセット（MNIST）で99%の精度を達成

深層学習



ImageNet Challenge

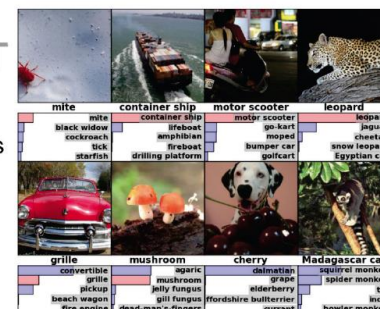
深層学習の発展：

1) 大規模のデータベース：ImageNet

ImageNet: 1,000カテゴリ,
約120万枚の訓練画像データ

IMAGENET

- 1,000 object classes (categories).
- Images:
 - 1.2 M train
 - 100k test.



2) High-performanceの計算機：GPU

3) 効率的な学習手法

Caffe

Chainer

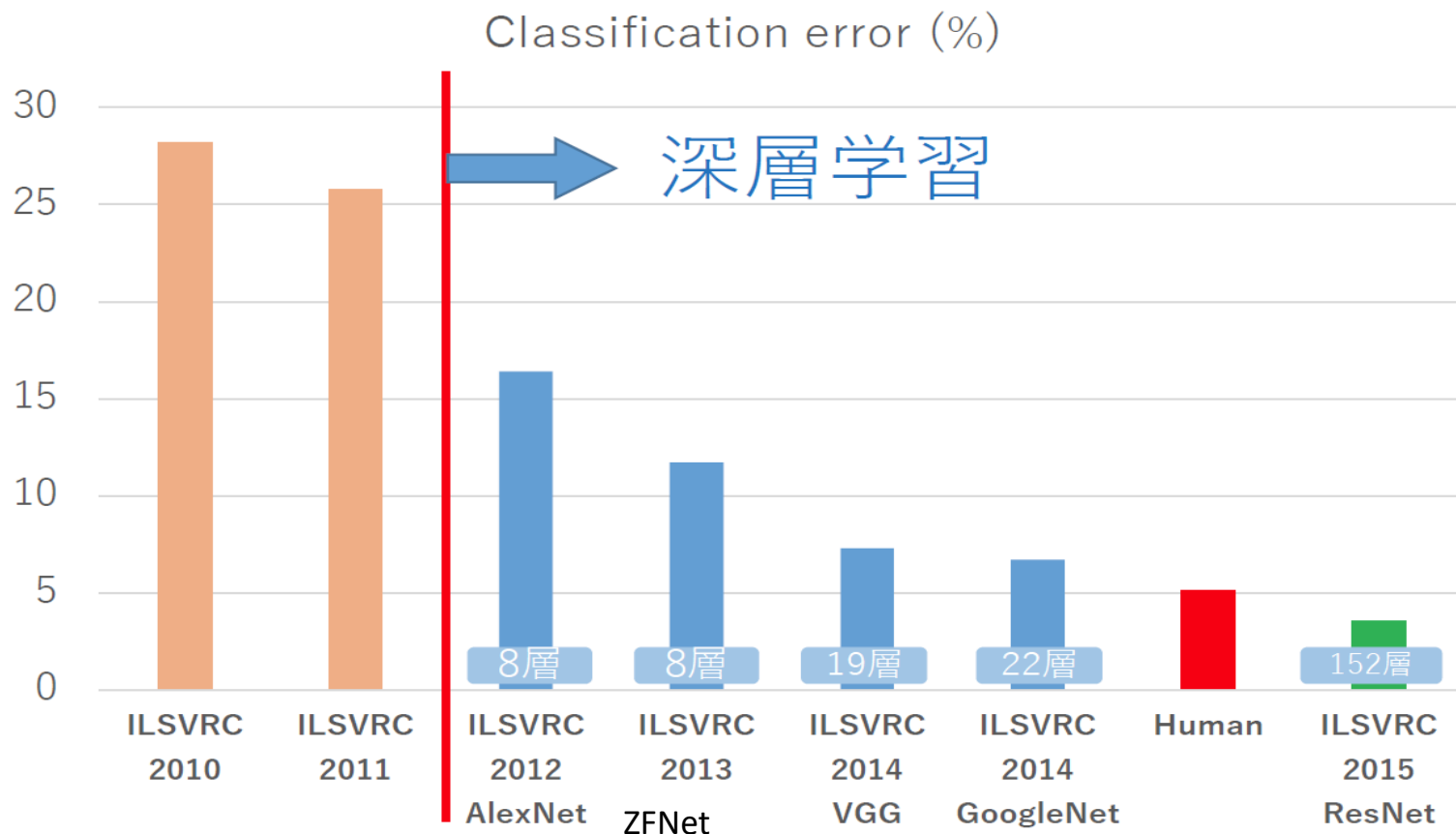
公開された様々なライブラリ

TensorFlow：2015年登場、Google製。産業界で人気

Keras：2015年登場、作者がGoogle社員。使いやすくて簡単。
TensorFlow 2に同梱され標準API化

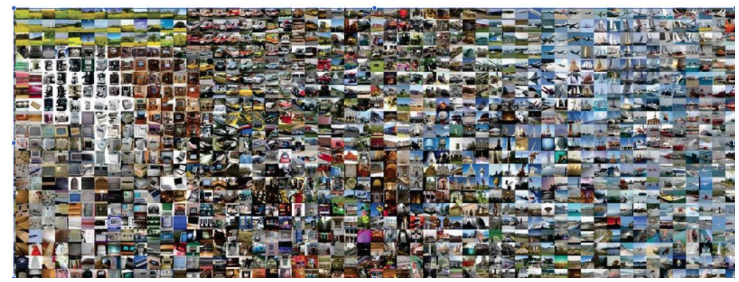
PyTorch：2016年登場、Facebook製。研究分野で人気

ImageNetデータにおける識別精度の変遷

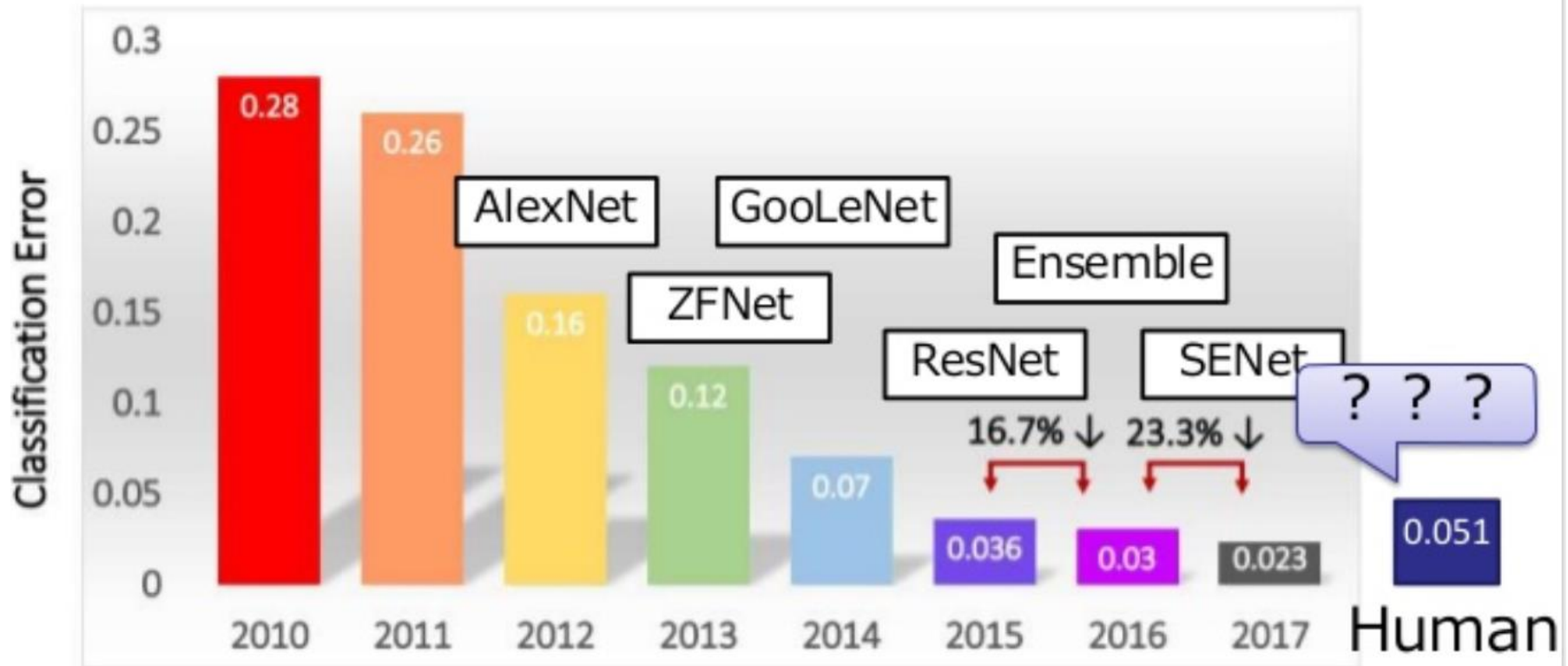


ImageNet: 21841クラス, 14,197,122枚の訓練画像データ
そのうち1000クラスでコンペティション

ImageNetデータにおける識別精度の変遷



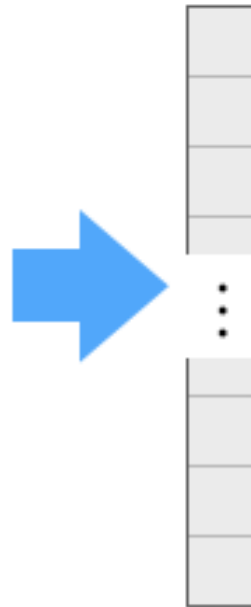
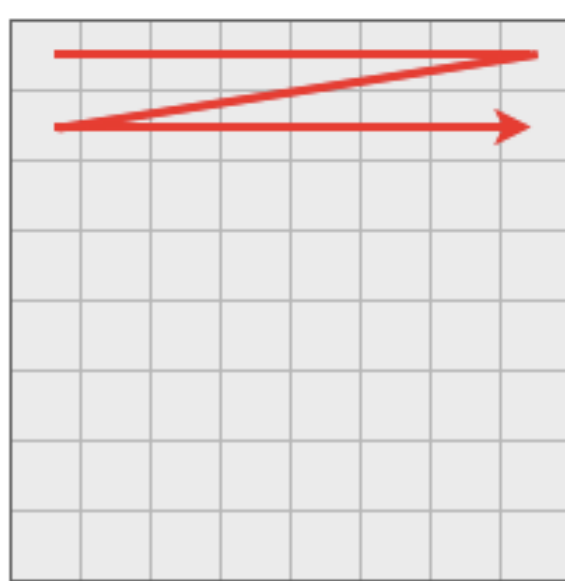
■ クラス分類タスクのエラー率 (top5 error) の推移



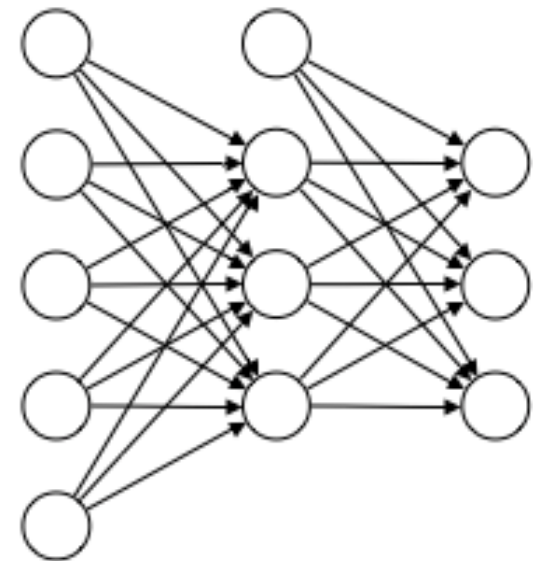
多層パーセプトロン

多層パーセプトロン：画像を対象にする場合は、強力ではない
ごく小さな画像なら学習できる

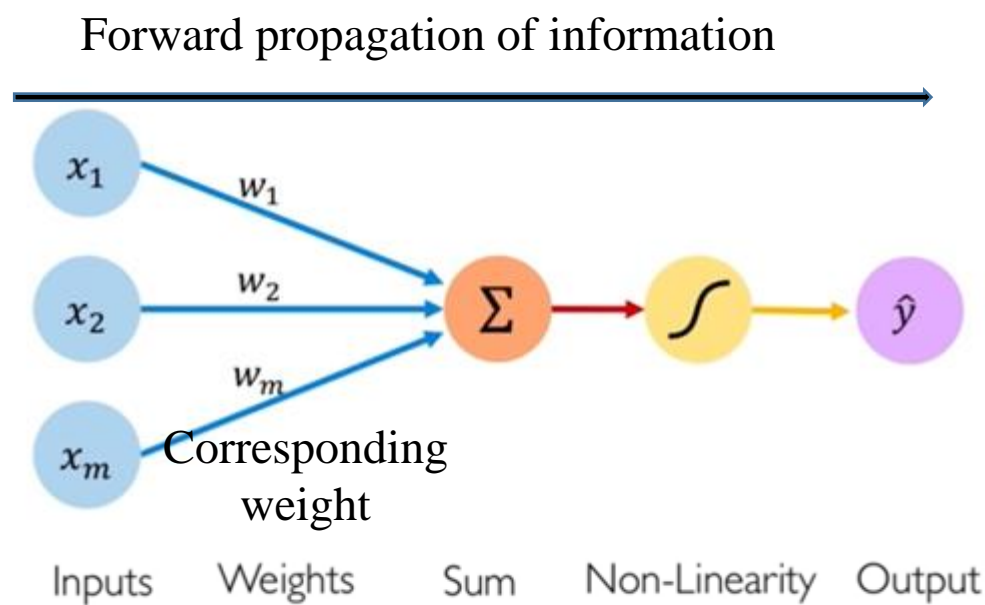
① 画像のピクセルの色濃度を一次元配列として読み込む



② 値を多層パーセプトロンに入力



多層パーセプトロン: Multi-layer Perceptron

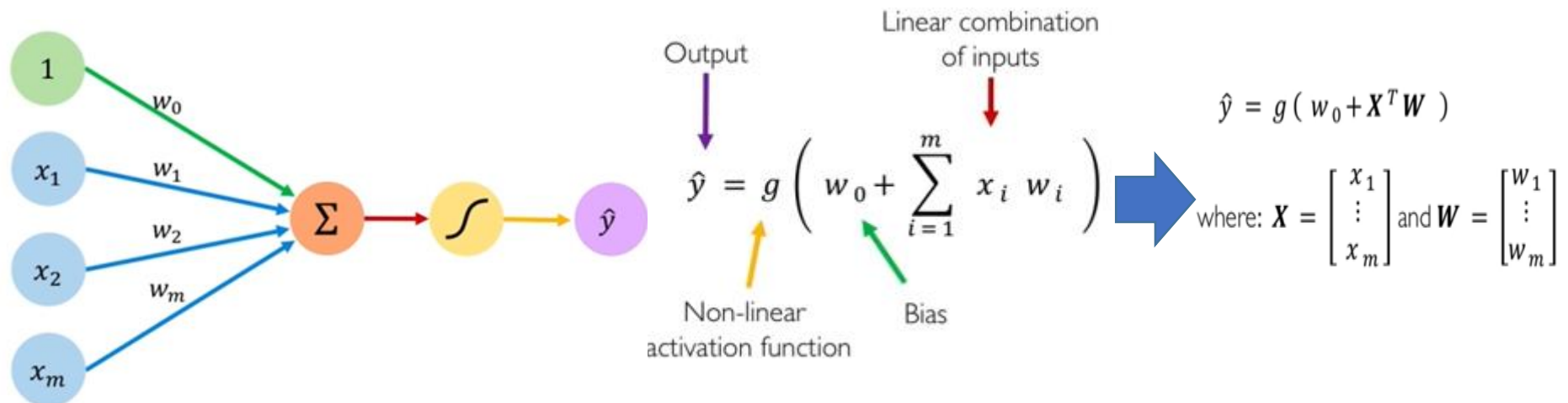


Linear combination of inputs

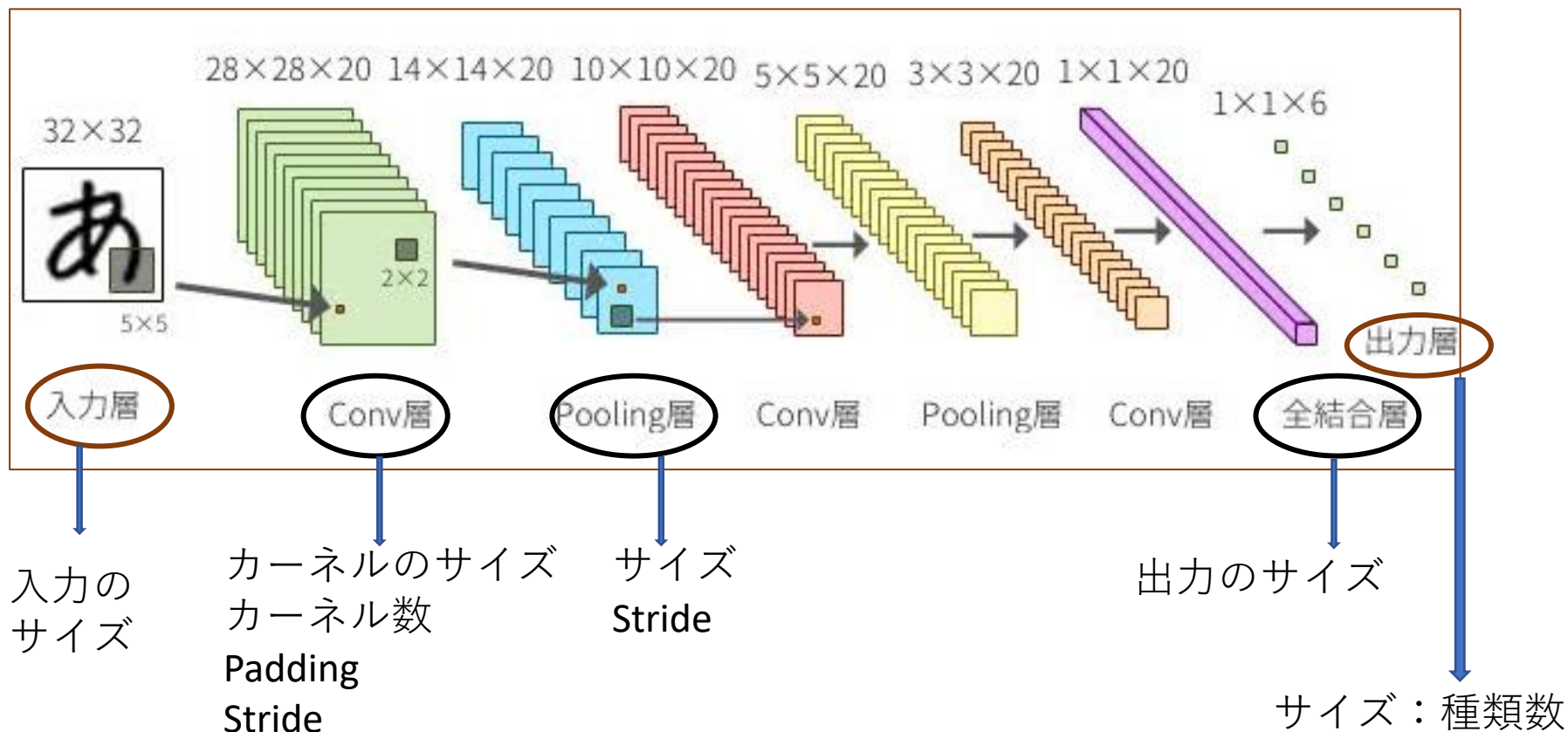
$$\hat{y} = g \left(\sum_{i=1}^m x_i w_i \right)$$

Output

Non-linear activation function



畳み込みネットワーク：簡単な構造



画像の畳込み演算

原画像



カーネル
(filter)

結果画像

| | | |
|----------------|---|--|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ | |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ | |
| | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ | |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ | |

カーネル
(filter)

結果画像

| | | |
|---|---|--|
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ | |
| Box blur (normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ | |
| Gaussian blur 3 × 3 (approximation) | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ | |
| Gaussian blur 5 × 5 (approximation) | $\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$ | |
| Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask) | $\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$ | |

畳み込み演算: 元の入力画像に対し、左上から右下まで要素ごとに掛け合わせていく。このようにしてフィルタをスライドさせていくことから **sliding window** と呼ばれることもある。

画像の各ピクセルの色濃度を、ピクセルの並び通りに二次元に配列として表現する。

(簡単のため、モノクロの画像として考える)

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


フィルタとなる二 次元の配列

| | | |
|-----|-----|-----|
| 0.2 | 0 | 0.4 |
| 0.1 | 0.3 | 0.2 |
| 0 | 0.1 | 0.1 |

1-① 画像の一番左上の部分にフィルタを重ね、
同じ位置の値同士ををかけ合わせる

| | | | | | | |
|-------------------|-------------------|-------------------|---|---|---|---|
| 0 $\times 0.2$ | 2 $\times 0$ | 2 $\times 0.4$ | 0 | 1 | 0 | 0 |
| 1 $\times 0.1$ | 8 $\times 0.3$ | 7 $\times 0.2$ | 5 | 2 | 3 | 6 |
| 3 $\times 0$ | 6 $\times 0.1$ | 9 $\times 0.1$ | 7 | 6 | 1 | 4 |
| 1 | 4 | 5 | 2 | 4 | 0 | 1 |
| 0 | 2 | 3 | 1 | 2 | 1 | 4 |

1-② かけ算の結果を足し合わせ、出力する



6.2

2-① フィルタを1ピクセル分だけ横にずらし、
同じように値同士を掛け合わせる

| | | | | | | |
|---|--------------|--------------|--------------|---|---|---|
| 0 | 2 | 2 | 0 | 1 | 0 | 0 |
| | $\times 0.2$ | $\times 0$ | $\times 0.4$ | | | |
| 1 | 8 | 7 | 5 | 2 | 3 | 6 |
| | $\times 0.1$ | $\times 0.3$ | $\times 0.2$ | | | |
| 3 | 6 | 9 | 7 | 6 | 1 | 4 |
| | $\times 0$ | $\times 0.1$ | $\times 0.1$ | | | |
| 1 | 4 | 5 | 2 | 4 | 0 | 1 |
| 0 | 2 | 3 | 1 | 2 | 1 | 4 |

2-② かけ算の結果を足し合わせ、出力する

| | | | |
|-----|-----|--|--|
| 6.2 | 5.9 | | |
| | | | |

畳み込みネットワークの特徴：

1. 合成性

CNNはそれぞれの構成要素を理解すると、パズルのように組み合わせてつくることができるようになる。

各層は次の層へと意味のあるデータを順に受渡していく。層が進むにつれて、ネットワークは**より高レベルな特徴を学習していける**。

2. 移動不変性

局所領域からフィルタを通して検出していくので物体の位置のズレに頑健になる。

つまり、特徴を検知する対象が入力データのどこにあっても検知することができる。これを移動不変性という

回転や拡大・縮小に対する不変性はどうなのだろう？ある程度は不変性を維持できるものの、それほど頑健ではないので、データ拡張でそういったデータを増やして学習するなど工夫が必要だ。（**水増し**）

ゼロパディング (zero padding)

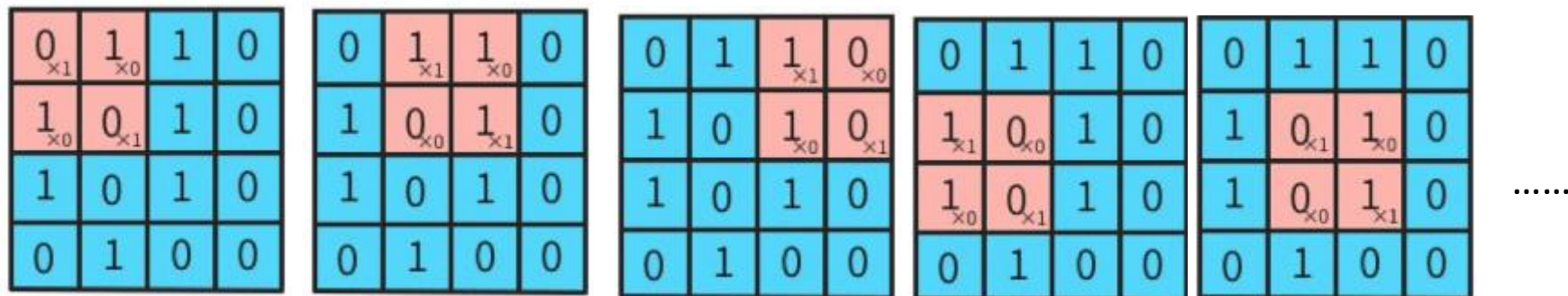
入力の特徴マップの周辺を0で埋める

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- 端のデータに対する畳み込み回数が増えるので端の特徴も考慮されるようになる
- 畳み込み演算の回数が増えるのでパラメーターの更新が多く実行される
- カーネルのサイズや、層の数を調整できる

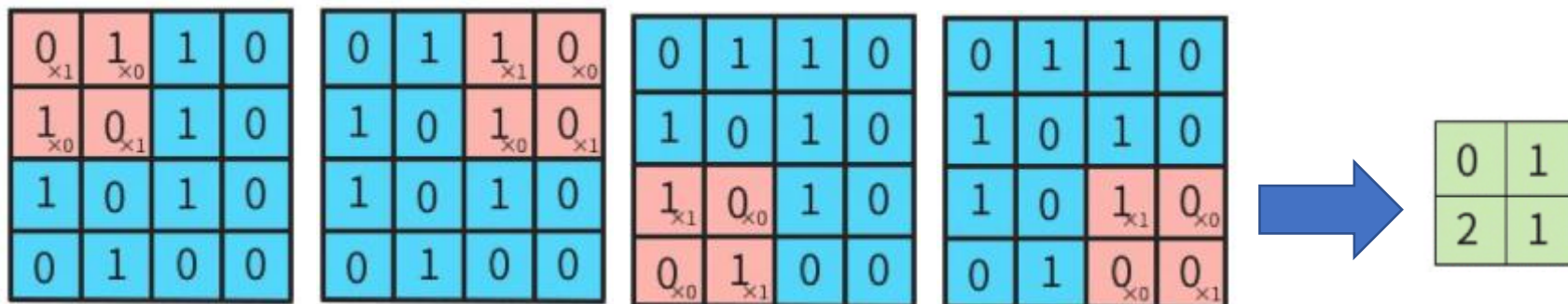
Stride: ストライド

Stride=1



| | | |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 1 | 1 |
| 2 | 0 | 1 |

Stride=2



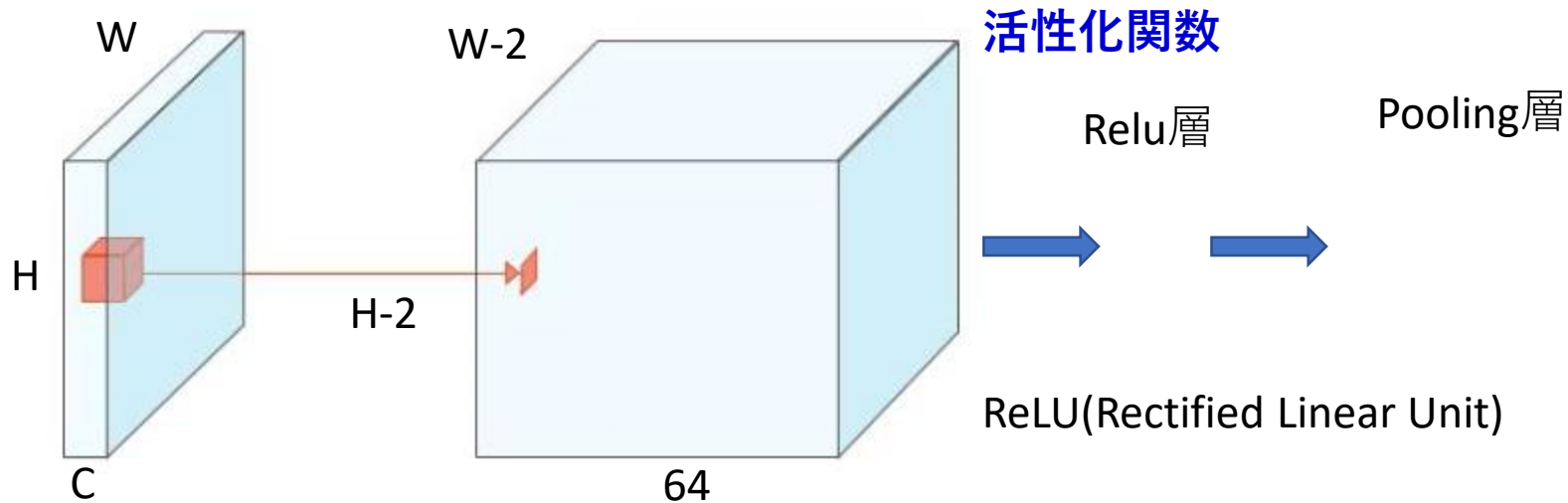
畳み込み層：

フィルタサイズ： $3 * 3 * C$

フィルタ数：64

Padding: 0

Stride: 1

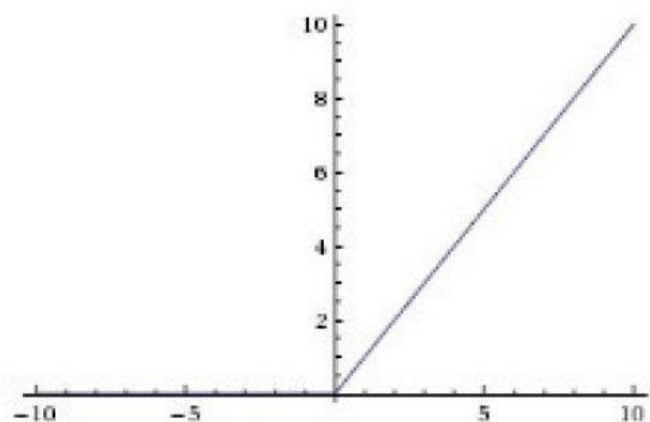


$\max(0, x)$ のような実装をする。
Convolution層ではこの活性化関数を使うことが多い。

活性化関数

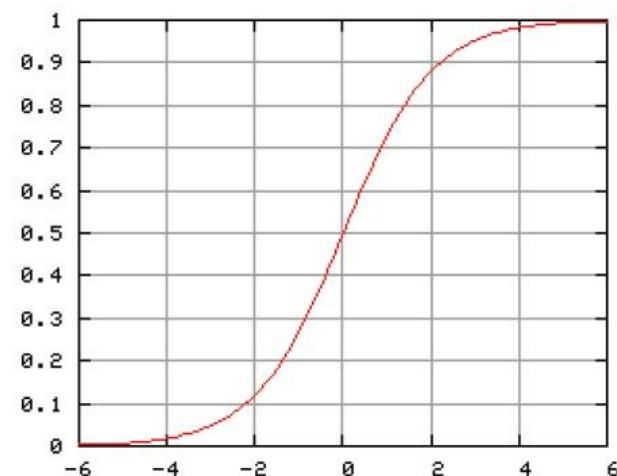
☆ReLU (Rectified Linear Unit)

$$h(u) = \max\{u, 0\}$$



シグモイド関数

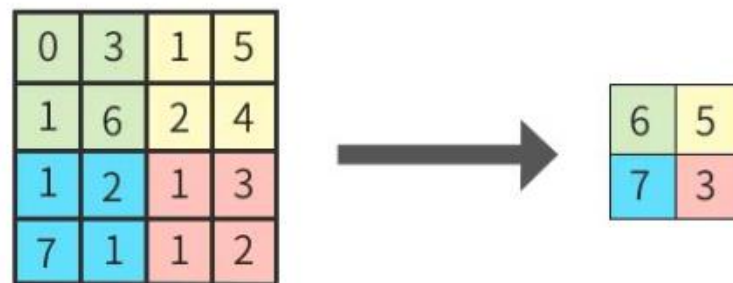
$$h(u) = \frac{1}{1 + e^{-u}}$$



Pooling層 : Pooling領域サイズ Stride

たいてい、Convolution層の後に適用される。入力データをより扱いやすい形に変形するために、情報を圧縮し、down samplingする

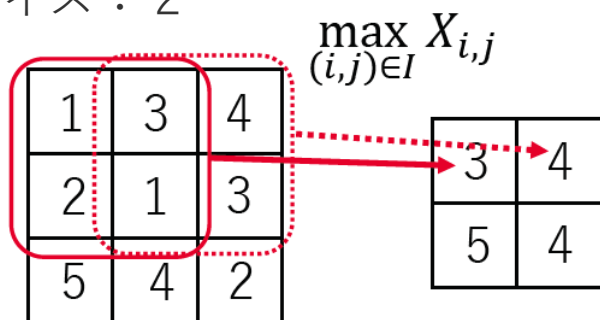
Pooling領域サイズ : 2
Stride : 2



Max Pooling

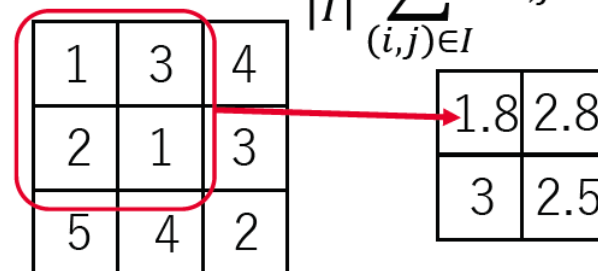
- 微小な位置変化に対して頑健となる
- ある程度過学習を抑制する
- 計算コストを下げる

Pooling領域サイズ : 2
Stride : 1



Max-プーリング

$$\frac{1}{|I|} \sum_{(i,j) \in I} X_{i,j}$$

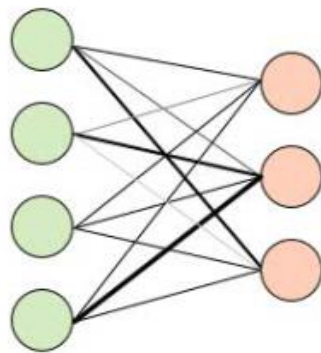


Average-プーリング

Fully Connected層

サイズ

一般的なニューラルネットワークでも見られるFully Connected層（全結合層）である。各層のユニットは次の層のユニットと**全て繋がっている**。

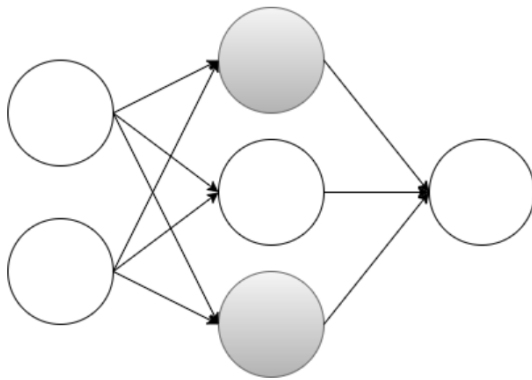


Fully Connected層

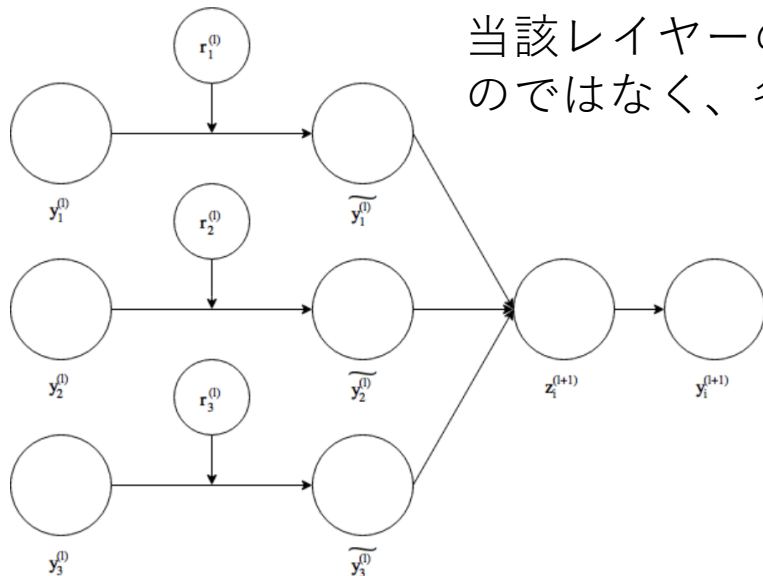
Fully Connected層は1次元のベクトルを入力値として、1次元のベクトルを出力する。つまり、空間的な位置情報を無視されてしまう

Dropout

ニューラルネットワークの学習時に、一定割合のノードを**不活性化させ**ながら学習を行うことで**過学習**を防ぎ（緩和し）、精度をあげるために手法



色のついたノードが不活性化されている



当該レイヤーのうちの指定割合だけノードを活性化させるのではなく、各ノードが活性化する**確率**が与えられる

数式的に：

$$r_j^{(l)} \sim \text{Bernoulli}(p)$$

$$\tilde{y}^{(l)} = r^{(l)} \times y^{(l)}$$

$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^l$$

$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

クラス分類CNNの流れ

- 1.画像入力
- 2.N次元のベクトルを答えとして生成 (N個のクラス)
- 3.Softmax関数で正規化
- 4.Cross Entropyで誤差計算
- 5.逆誤差伝播法で誤差最小化

$$\text{Softmax}(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

$$\text{Cross Entropy}(\mathbf{x}, \mathbf{y}) = - \sum_{i=0}^n y_i \log x_i$$

クラス分類CNNの流れ

- 1.画像入力
- 2.N次元のベクトルを答えとして生成 (N個のクラス)
- 3.Softmax関数で正規化
- 4.Cross Entropyで誤差計算
- 5.逆誤差伝播法で誤差最小化

$$\text{Softmax}(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

$$\text{Cross Entropy}(\mathbf{x}, \mathbf{y}) = - \sum_{i=0}^n y_i \log x_i$$

クラス分類CNNの流れ

- 1.画像入力
- 2.N次元のベクトルを答えとして生成 (N個のクラス)
- 3.Softmax関数で正規化
- 4.Cross Entropyで誤差計算
- 5.逆誤差伝播法で誤差最小化

$$\text{Softmax}(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

$$\text{Cross Entropy}(\mathbf{x}, \mathbf{y}) = - \sum_{i=0}^n y_i \log x_i$$

確率ベクトル

分類CNNの損失関数：最小化

経験損失（訓練誤差）

$$L(W) = \sum_{i=1}^n \ell(y_i, f(x_i, W))$$

$$\ell(y, y') = (y - y')^2 \quad \text{二乗損失（回帰）} \quad (y, y' \in \mathbb{R})$$

$$\ell(y, y') = - \sum_{k=1}^K y_k \log(y'_k) \quad \text{Cross-entropy損失（多値判別）}$$

$(y_k \in \{0,1\}, y'_k \in [0,1], \text{ともに和が} 1)$

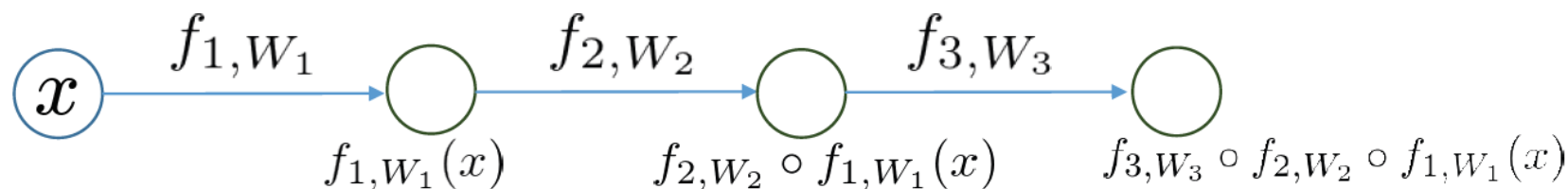
$$\min_W L(W)$$

$$W^t = W^{t-1} - \eta \partial_W L(W)$$

- 基本的には確率的勾配降下法（SGD）で最適化を実行
- AdaGrad, Adam, Natural gradientといった方法で高速化

微分はどうやって求める？ → 誤差逆伝搬法

損失関数最小化：誤差逆伝搬法



$$\text{例： } f_{1,W_1}(x) = h(W_1 x)$$

合成関数

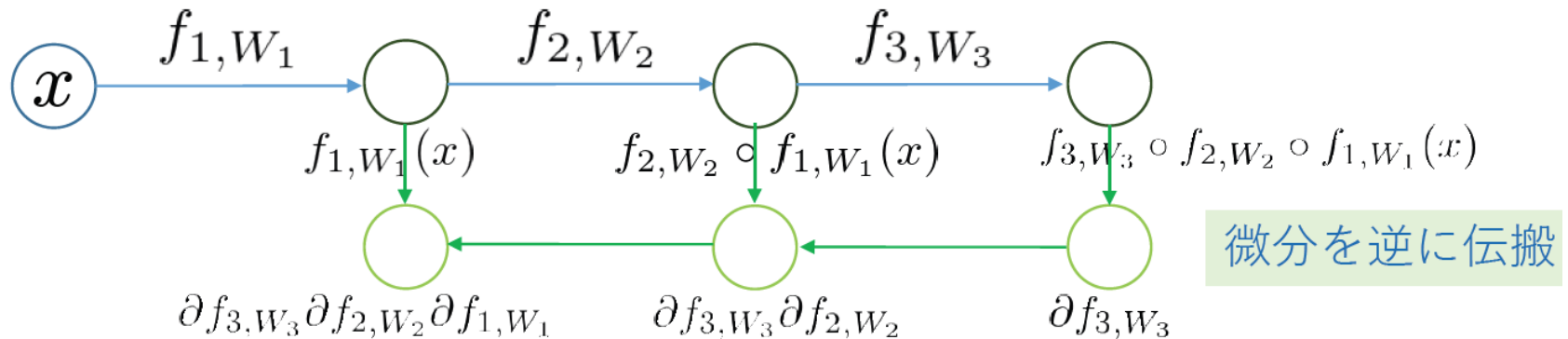
$$\begin{aligned} f(x; W) &= f_{3,W_3}(f_{2,W_2}(f_{1,W_1}(x))) \\ &= f_{3,W_3} \circ f_{2,W_2} \circ f_{1,W_1}(x) \end{aligned}$$

合成関数の微分

$$\frac{\partial f}{\partial W_1}(x) = \frac{\partial f_{3,W_3}}{\partial f_{2,W_2}} \frac{\partial f_{2,W_2}}{\partial f_{1,W_1}} \frac{\partial f_{1,W_1}}{\partial W_1}(x)$$

損失関数最小化：誤差逆伝搬法

20



連鎖律を用いて微分を伝搬

$$\frac{\partial f}{\partial W_3}(x) = \frac{\partial f_{3,W_3}}{\partial W_3} (f_{2,W_2} \circ f_{3,W_3}(x))$$

$$\frac{\partial f}{\partial W_2}(x) = \frac{\partial f_{3,W_3}}{\partial f_{2,W_2}} \frac{\partial f_{2,W_2}}{\partial W_2} (f_{3,W_3}(x))$$

$$\frac{\partial f}{\partial W_1}(x) = \frac{\partial f_{3,W_3}}{\partial f_{2,W_2}} \frac{\partial f_{2,W_2}}{\partial f_{1,W_1}} \frac{\partial f_{1,W_1}}{\partial W_1}(x)$$

パラメータによる微分と入力による微分は違うが、情報をシェアできる。

$f_{1,W}(x) = h(Wx)$ の場合

$$u = Wx$$

$$\frac{\partial f_{1,W}}{\partial W_{ij}}(x) = \frac{\partial h}{\partial u_i}(u) x_j$$

$$\frac{\partial f_{1,W_1}}{\partial x_j}(x) = \sum_i \frac{\partial h}{\partial u_i}(u) W_{ij}$$

損失関数最小化：誤差逆伝搬法

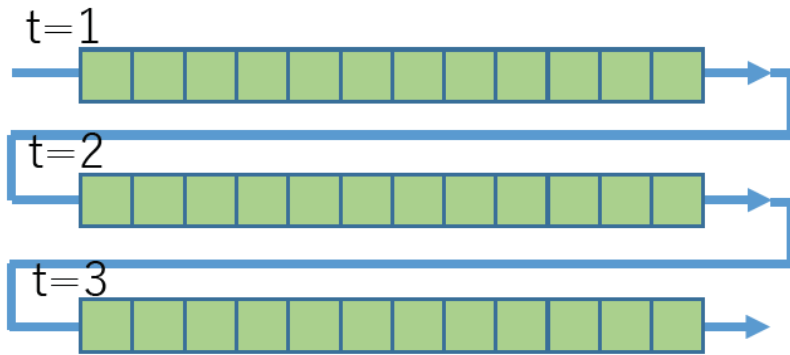
確率的勾配降下法(確率的勾配降下法：SGD)

沢山データがあるときに強力

$$\min_W \frac{1}{n} \underbrace{\sum_{i=1}^n \ell(z_i, W)}_{\text{重い}}$$

大きな問題を分割して個別に処理

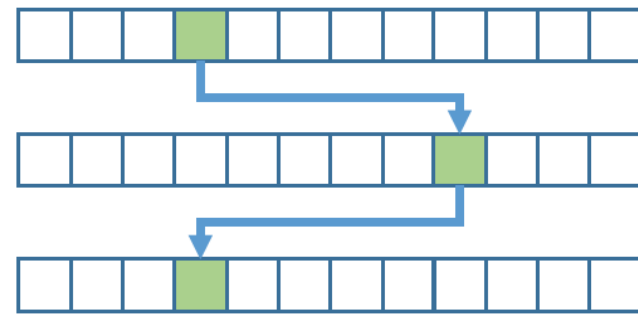
普通の勾配降下法：



$$W^t = W^{t-1} - \alpha \frac{1}{n} \sum_{i=1}^n \nabla \ell(z_i, W)$$

確率的勾配降下法：

毎回の更新でデータを一つ(または少量)しか見ない



$$W^t = W^{t-1} - \alpha \nabla \ell(z_i, W)$$

損失関数最小化：誤差逆伝搬法

確率的勾配降下法(確率的勾配降下法：SGD)

沢山データがあるときに強力

$$\min_W \frac{1}{n} \underbrace{\sum_{i=1}^n \ell(z_i, W)}_{\text{重い}}$$

大きな問題を分割して個別に処理

- ランダムに一つのデータ z_{i_t} を観測.
- 選択した一つのデータで勾配を計算：

$$g_t = \nabla_W \ell(z_i, W^{(t-1)})$$

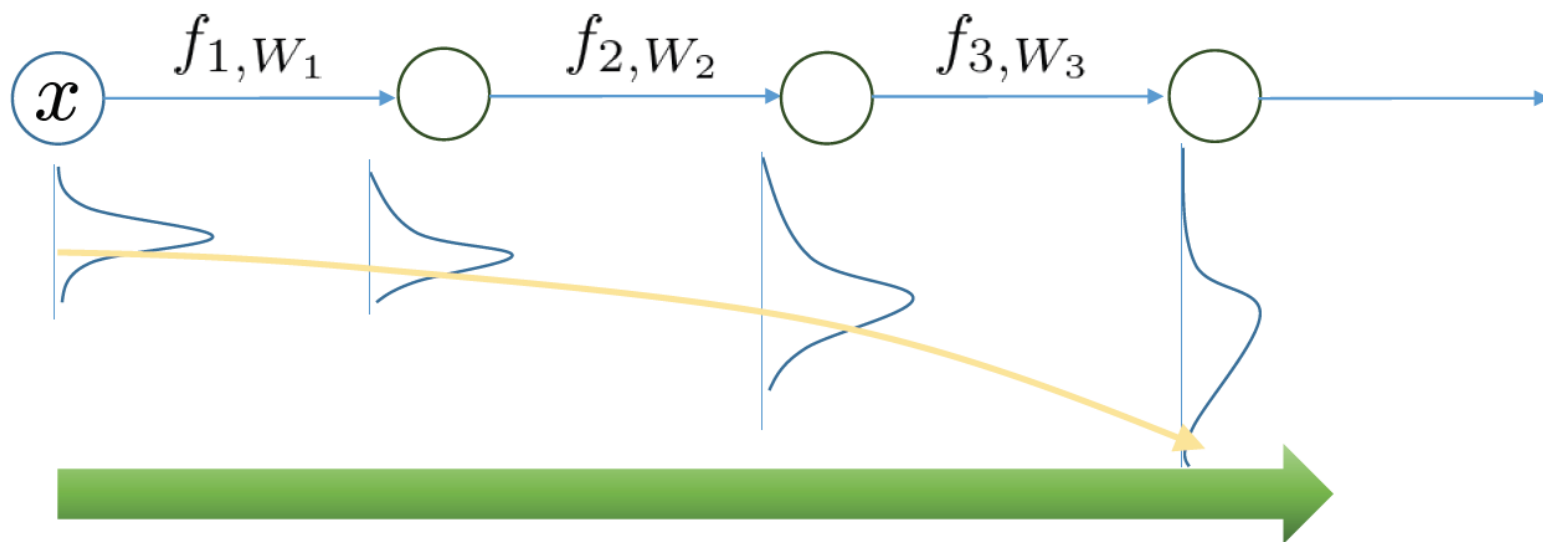
← $O(1)$ の計算量

- 勾配方向へ更新（近接勾配法と同じ更新式）：

$$W^{(t)} = W^{(t-1)} - \alpha g_t$$

理論的にも実験的にもトータルの計算量で得ることが知られている

バッチ正規化(batch normalization)



- 深くなるにつれ、平均と分散が発散or縮小してゆき、学習が安定しない。
 - バッチ正規化はこの揺れによる自由度を解消し、学習を安定化させる。
- ※ReLUを用いている限りスケール不変なのでスケールは固定したほうが良い。

各座標ごとに
平均と分散を正規化

$$\hat{x} \leftarrow \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

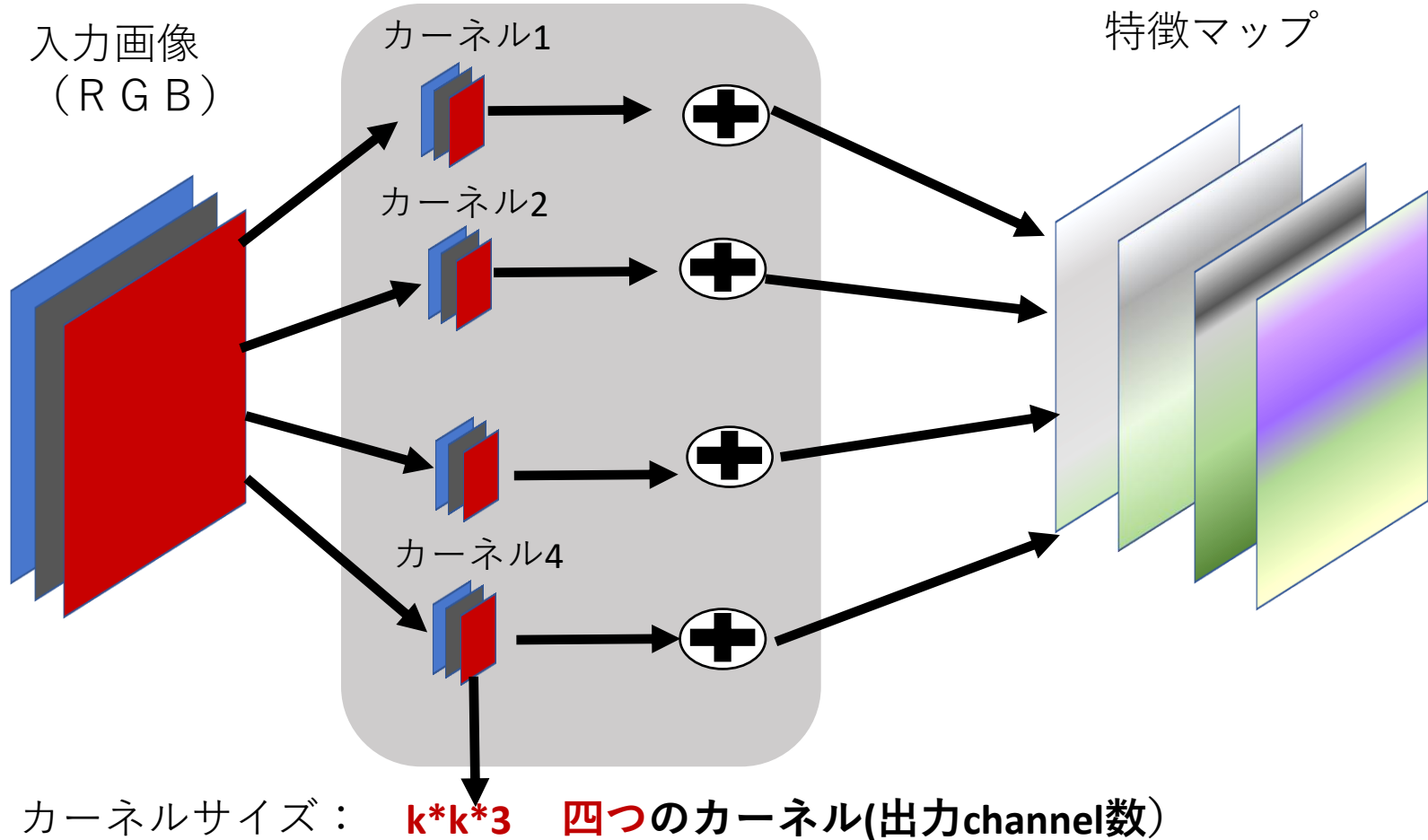
μ_B : ミニバッチ内の平均, σ_B^2 : ミニバッチ内の分散

その他の演算

- 1) グループ畳み込み層 (Group Convolution)
- 2) Depthwise畳み込み層
- 3) Pointwise畳み込み層
- 4) 逆畳み込み
(Deconvolution or Transpose畳み込み)

古典的な畳み込み

畳み込み層

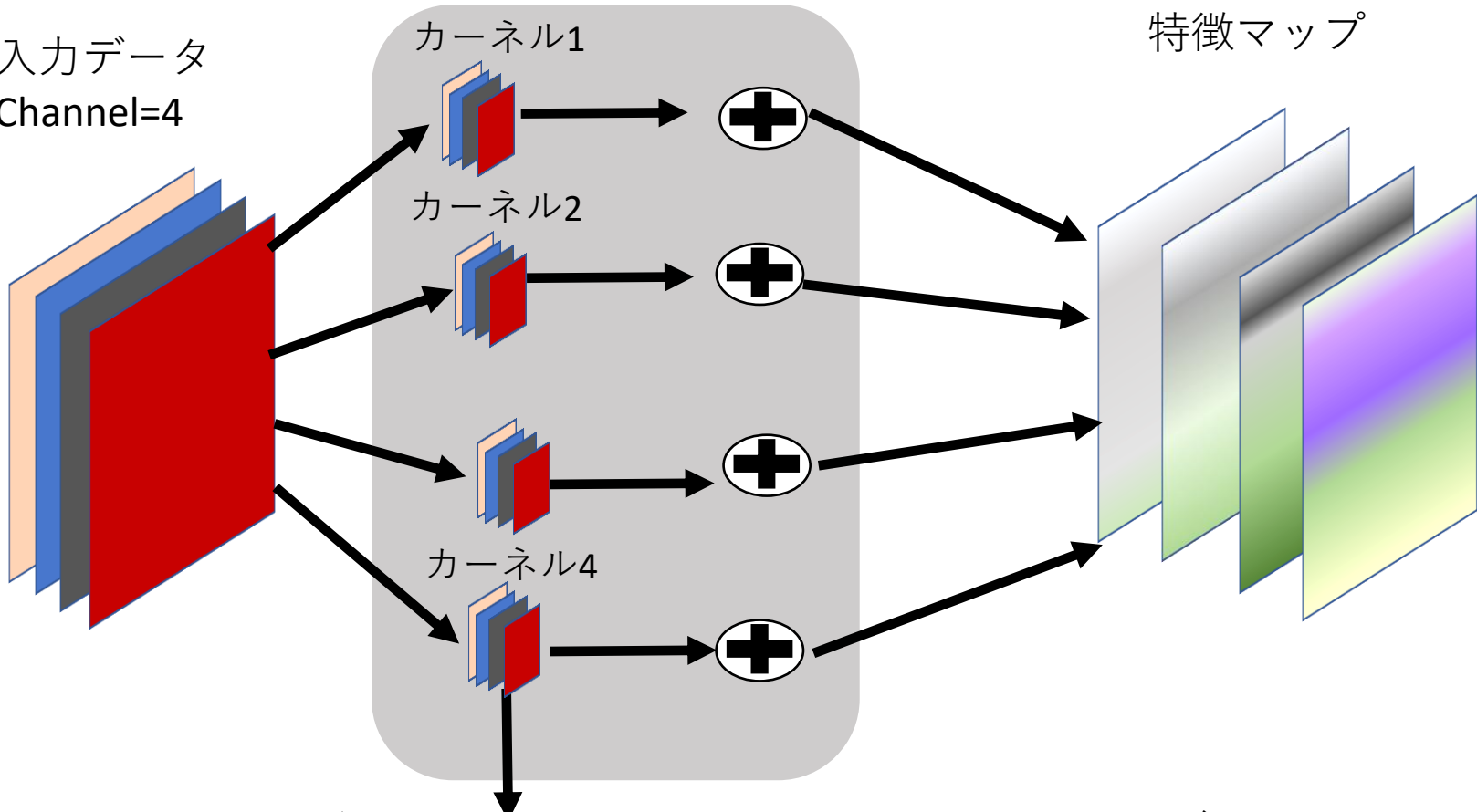


パラメータ数: $4 \times k \times k \times 3$

古典的な畳み込み

畳み込み層

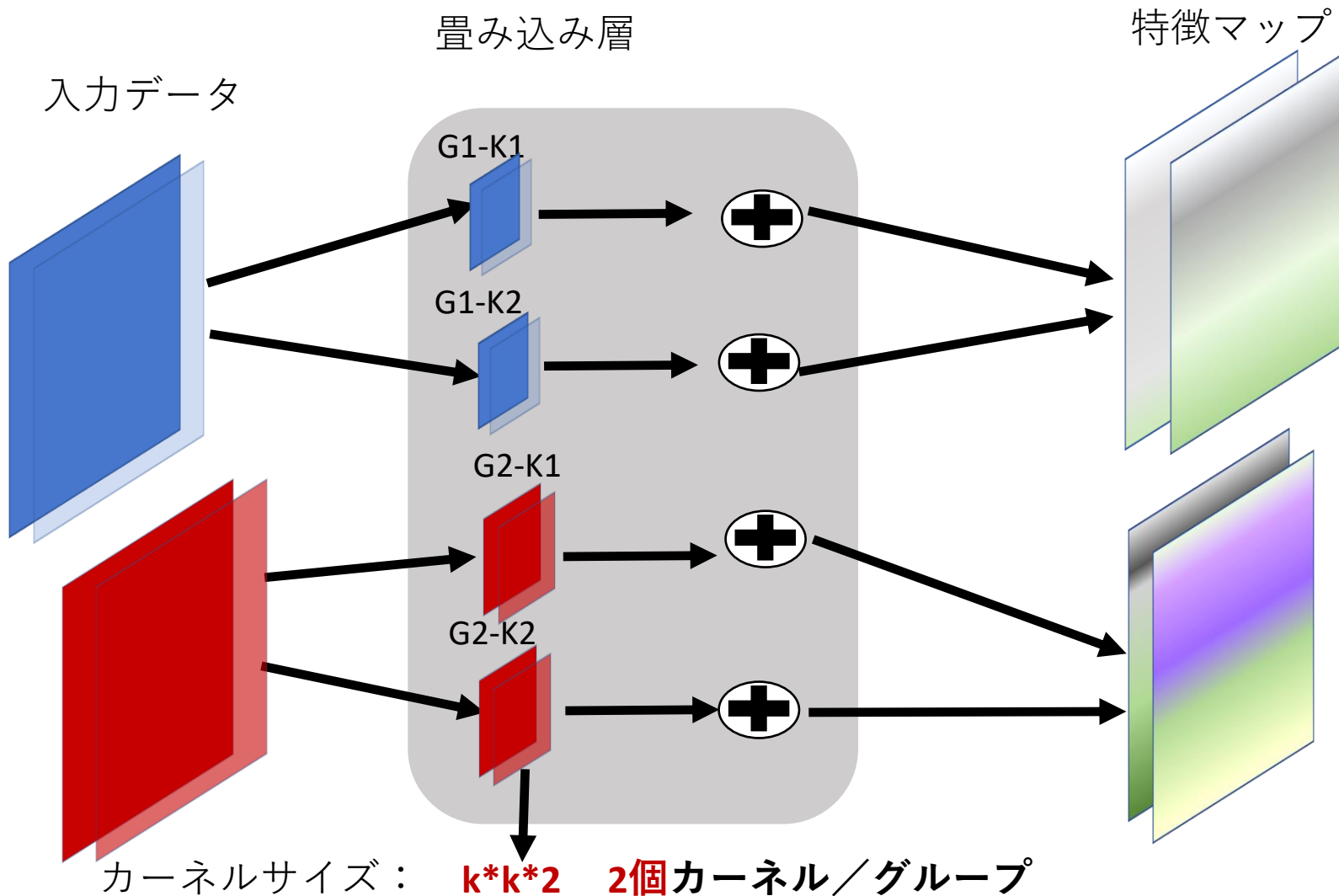
入力データ
Channel=4



カーネルサイズ： $k \times k \times 4$ 四つのカーネル (出力channel数)

パラメータ数： $4 \times k \times k \times 4$

Group畳み込み

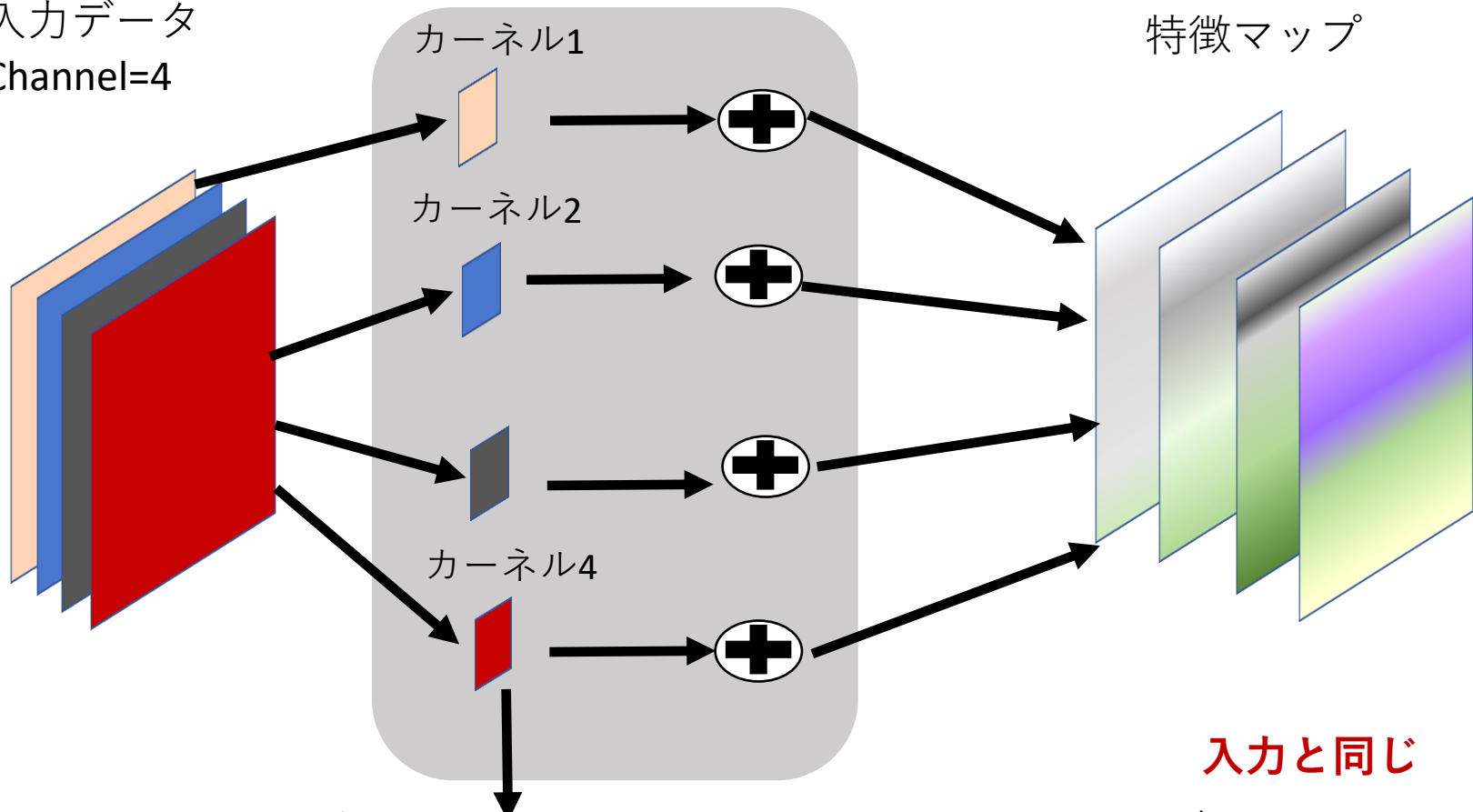


パラメータ数: $2 \times 2 \times k \times k \times 2$

DepthWise畳み込み

畳み込み層

入力データ
Channel=4



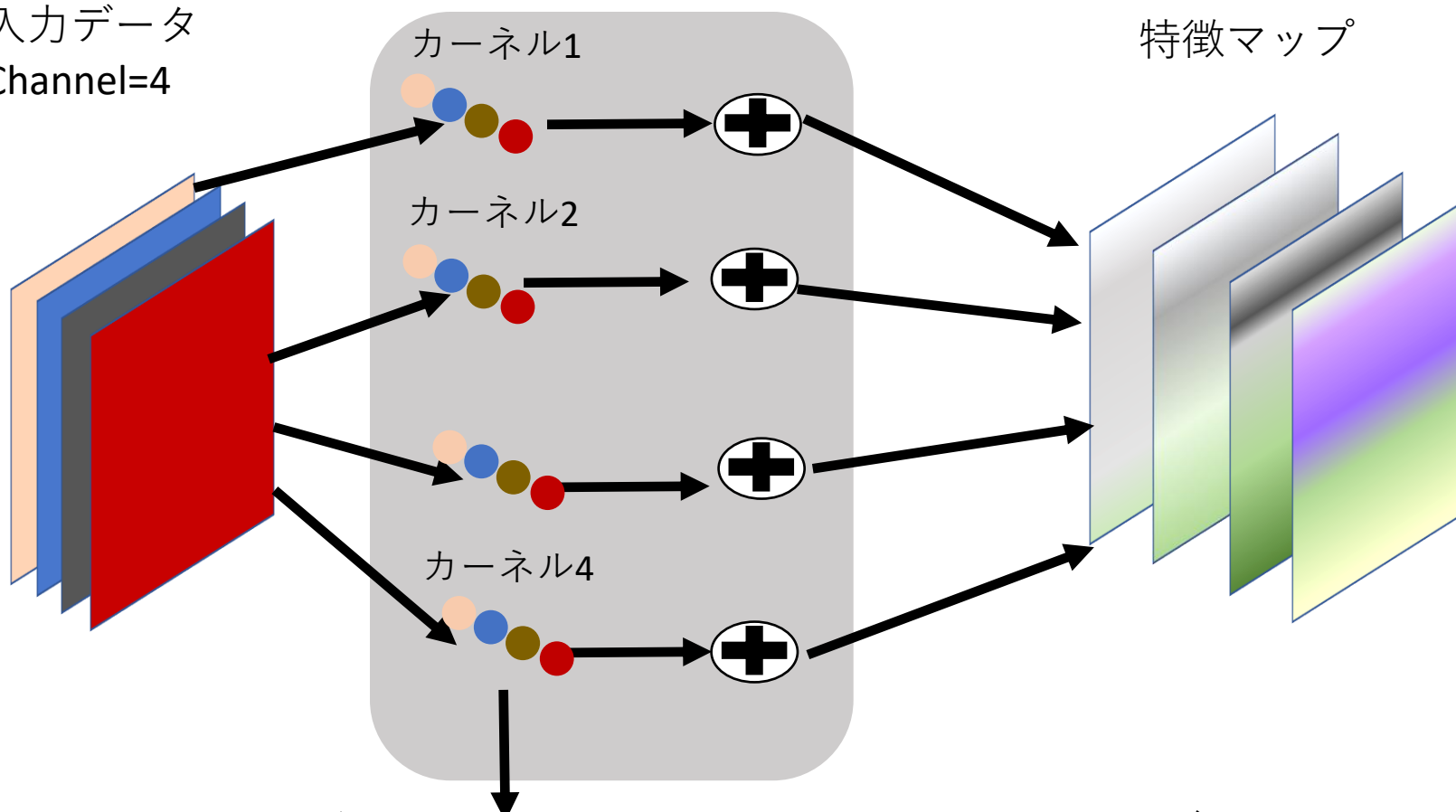
カーネルサイズ： $k*k*1$ 四つのカーネル (出力channel数)

パラメータ数： $4 * k*k*1$

Point Wise 畳み込み

畳み込み層

入力データ
Channel=4

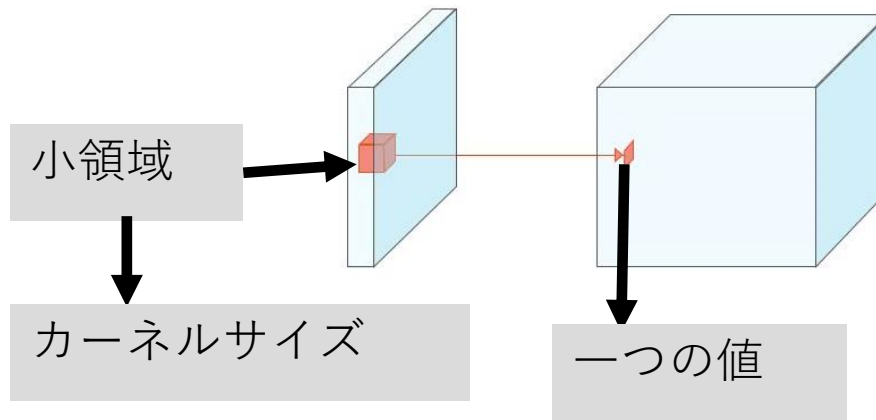


カーネルサイズ： $1*1*4$ 四つのカーネル (出力channel数)

パラメータ数： $4*1*1*4$

逆畳み込み: Deconvolution

畳み込み層: 入力データの小さい領域を集約して、一つの値を計算



Down-Sampling

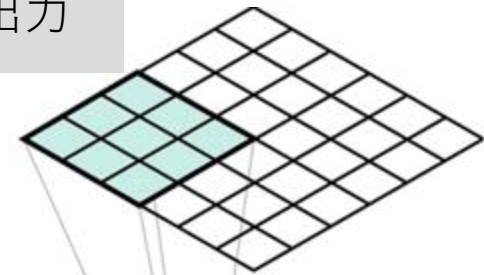
strideの値を変更することで特徴マップサイズを小さくする

逆畳み込み: 入力データの一つの要素をカーネルサイズの出力領域になる

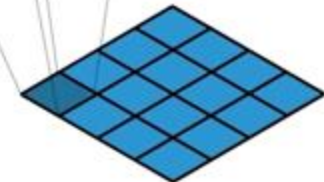
UP-Sampling (補間)

strideの値を変更することで特徴マップサイズを大きくする

出力



入力

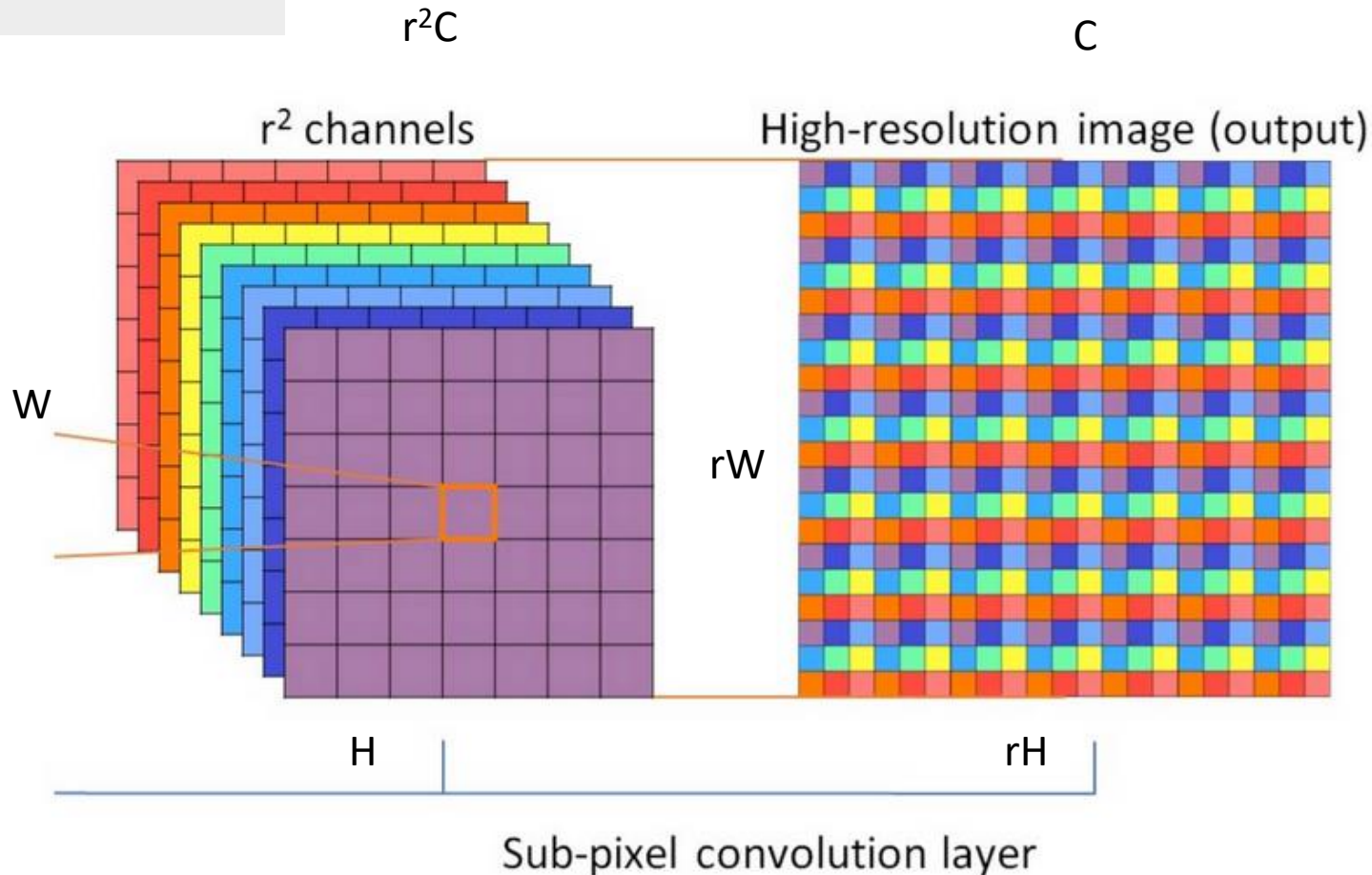


逆畳み込み: Deconvolution

Pixel Shufflerを使って特徴マップを大きくする

r倍を拡大

Sub-Pixel Convolution



Skip Connection

- 1) CNNは層を深くすることで精度が向上する傾向がある
- 2) 層を重ねることで高度で複雑な特徴を抽出できる

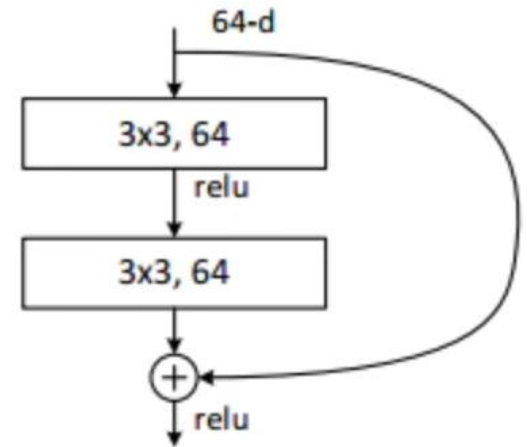
Deepすぎると学習進まない

ネットワークの浅い層（前半）の特徴マップを深い層（後半：サイズの等しい）特徴マップに連結する手法で

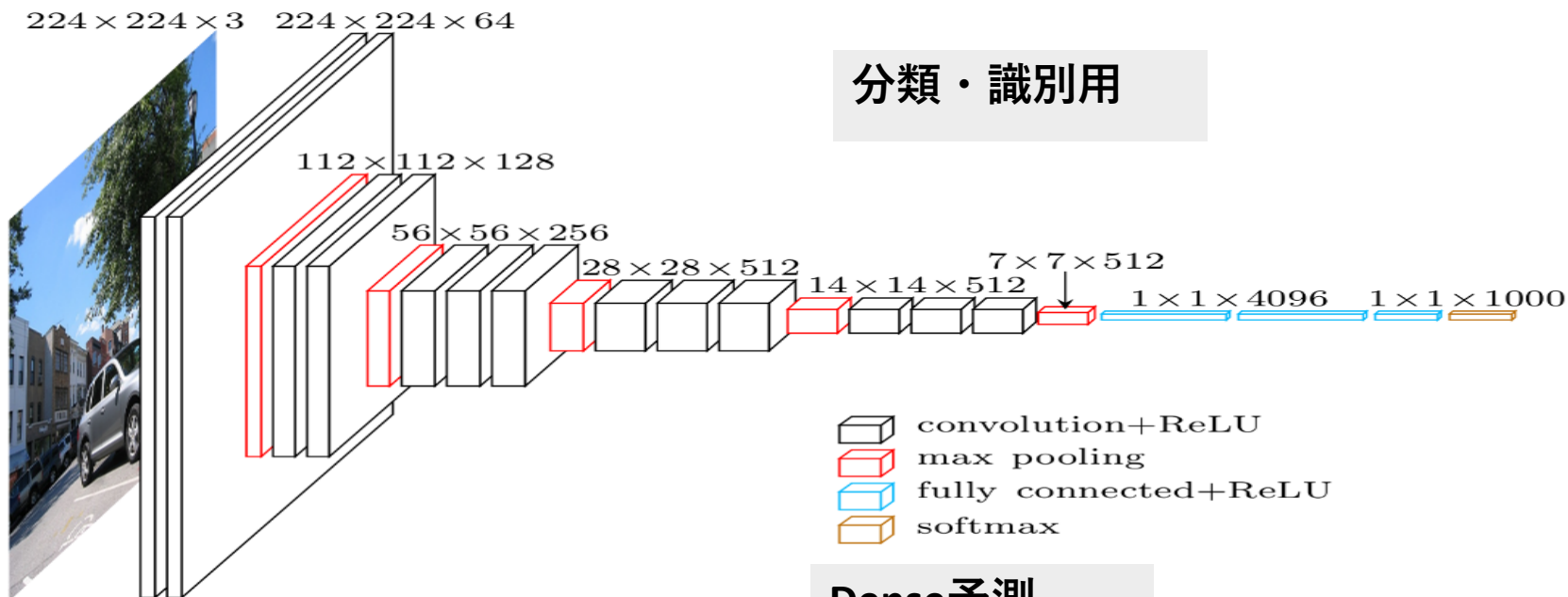
浅い層： 画像の局所的特徴
深い層： 画像の全体的特徴

Skip-Connection： 画像の全体的特徴と局所的特徴を統合して学習

物体の境界部分の曖昧さを低減する効果が期待できる

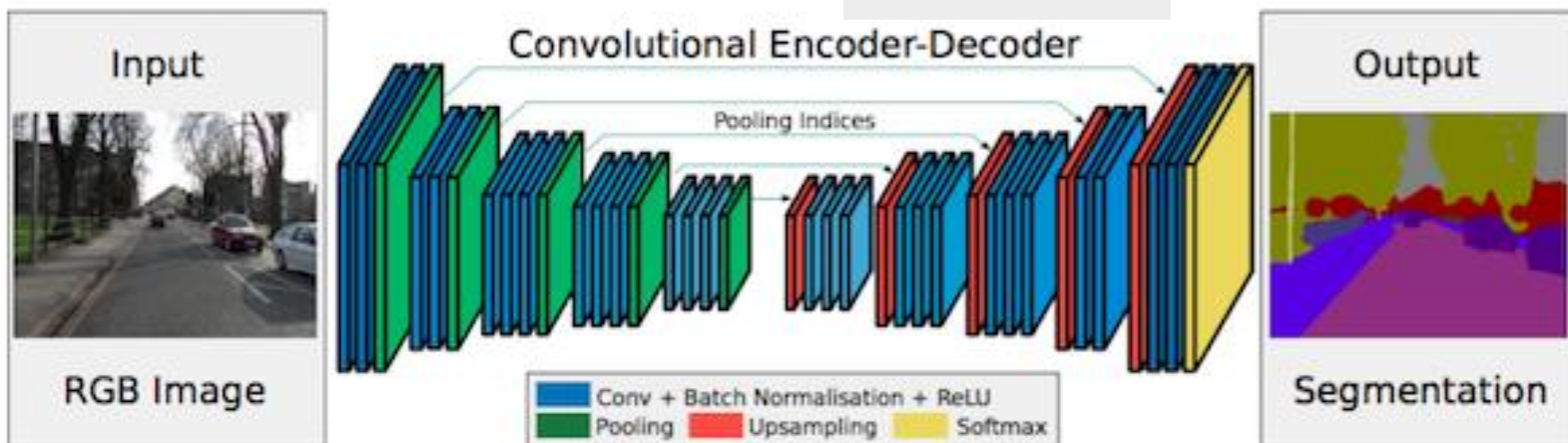


ネットワーク構造例



分類・識別用

Dense予測



まとめ

- **深層学習の発展**
- 畳込みネットワークの構造
- 基本的な演算
 - 1) 畳込み層、Pooling層、全結合層、Batch Normalization
活性化関数
 - 2) グループ畳込み、Depthwise, Pointwise畳込み、逆畳込み
(Pixel Shuffler)
 - 3) Skip Connection
- ネットワーク構造例