深層学習 Part 3

東京海洋大学

竹縄知之

目次

1 畳み込みニューラルネットワーク 5

1.1 畳み込み演算 6
1.2 実際の適用のための拡張 9
1.3 畳み込み演算（チャンネル，ミニバッチあり） 13
1.4 畳み込み演算に関する演習 19
1.5 プーリング 20
1.6 プーリング演算に関する演習 23

2 畳み込みニューラルネットワークの実装 24

2.1 畳み込み層 25
2.2 畳み込み層に関する演習 29
2.3 プーリング層 30
2.4 プーリング層に関する演習 31
2.5 畳み込みニューラルネットワークの実装に関する演習 32
2.6 GPU の利用に関する演習 33

3 CNN の利用 34

3.1 データの種類 35

3.2 データ増大 36
3.3 データ増大に関する演習 38
3.4 特徴量の転移 39

4 CNN の発展 41
4.1 VGG および GooglNet 43
4.2 Residual Networks 45
4.3 ResNet に関する演習 49

5 生成モデル 50
5.1 生成モデルと識別モデル 51
5.2 オートエンコーダ 52
5.3 逆畳み込み（転置畳み込み）と畳み込みオートエンコーダ 53
5.4 U-Net 56
5.5 オートエンコーダ（U-Net）によるセグメンテーションに関する演習 57
5.6 GAN 58
5.7 DCGAN 63
5.8 Conditionnal GAN 64
5.9 pix2pix 65
5.10 pix2pix に関する演習 67

6 一般物体検出—画像の局在化・検知・セグメンテーション 68

6.1 Faster R-CNN 69
6.2 YOLO と SSD 72
6.3 セマンティック・セグメンテーション 74
6.4 SegNet 77
6.5 Mask R-CNN に関する演習 78

1　畳み込みニューラルネットワーク

本節では，深層学習の最も基本的な手法である畳み込みニューラルネットワーク（Convolutional Neural Networks, CNN）について学びます．

CNN は 1 次元や 2 次元，ときには 3 次元の格子状のデータに対して有効なニューラルネットワークであり，主として「畳み込み操作」と「プーリング操作」によって構成されます．本節では特によく用いられる 2 次元の画像データを例として説明します．

CNN は 2012 年に行われたの画像認識コンペティション ImageNet Large Scale Visual Recognition Competition (ILSVRC) において，AlexNet と呼ばれる CNN を用いた手法が，それまでのサポートベクターマシンを用いたアプローチに大差をつけて優勝し，深層学習が一躍注目されるきっかけになりました．

1.1　畳み込み演算

データーつ，チャンネルなしの **2** 次元データのとき

畳み込み演算（または畳み込み処理）とは，例えば，

$$\text{入力 } X = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \quad \text{フィルター } F = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 2 & 0 & 1 \\ \hline -1 & -1 & 0 \\ \hline \end{array}$$

(グレーの部分は以下の y_{00} を計算するときに使うエリア)，に対して，出力 Y を

$$\begin{aligned} y_{00} &= (1,2,3,5,6,7,9,10,11)\cdot(1,1,1,2,0,1,-1,-1,0) \\ &= 1+2+3+10+7-9-10=4 \\ y_{01} &= (2,3,4,6,7,8,10,11,12)\cdot(1,1,1,2,0,1,-1,-1,0) \\ &= 2+3+4+12+8-10-11=8 \\ y_{10} &= (5,6,7,9,10,11,13,14,15)\cdot(1,1,1,2,0,1,-1,-1,0) \\ &= 5+6+7+18+11-13-14=20 \\ y_{01} &= (6,7,8,10,11,12,12,15,16)\cdot(1,1,1,2,0,1,-1,-1,0) \\ &= 6+7+8+20+12-14-15=24 \end{aligned}$$

$$\text{より } Y = \begin{array}{|c|c|}\hline 4 & 8 \\ \hline 20 & 24 \\ \hline \end{array}$$

と計算します.

つまり,

$$\text{入力 } X = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \quad \text{フィルター } F = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 2 & 0 & 1 \\ \hline -1 & -1 & 0 \\ \hline \end{array} \Longrightarrow Y = \begin{array}{|c|c|}\hline 4 & 8 \\ \hline 20 & 24 \\ \hline \end{array}$$

において,

- y_{00} はフィルターと X の左上隅のそれと同じ形の部分の「内積」
- y_{01} は X からとる部分を右に 1 ずらしたもの
- y_{10} は下に 1 ずらしたもの
- y_{11} は下に 1 右に 1 ずらしたもの

とする計算法です.

畳み込み演算を

$$\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \circledast \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 2 & 0 & 1 \\ \hline -1 & -1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 4 & 8 \\ \hline 20 & 24 \\ \hline \end{array}$$

と $\circledast$ という記号で表します.

形式的な定義と出力のサイズ

入力データ X のサイズを $\mathrm{H} \times \mathrm{W}$，フィルター F のサイズを $\mathrm{FH} \times \mathrm{FW}$ とするとき，出力データ Y を $(\mathrm{OH},\ \mathrm{OW}) = \big((\mathrm{H} - \mathrm{FH} + 1), (\mathrm{W} - \mathrm{FW} + 1)\big)$ 型の行列で，その (o_h, o_w) 成分が

$$Y[o_h, o_w] = \sum_{f_h=0}^{\mathrm{FH}-1} \sum_{f_w=0}^{\mathrm{FW}-1} X[o_h + f_h, o_w + f_w]\, F[f_h, f_w]$$

であるものと定めます．ただし，配列 A にたいして $A[i, j] = a_{i,j}$ はその (i, j) 成分を表すこととします．

畳み込み演算の特徴

結合が疎であること：　隣り合う層のニューロン同士は繋がっていない方が多い
パラメータ共有：　フィルターは位置に依らず同じものを使う

等により，画像の情報を段階的に取り出すことが出来る点にあります．

1.2　実際の適用のための拡張

　実際にニューラルネットワークで畳み込み演算を用いるときは，基本操作を若干拡張したものを使用します.

バイアス

出力結果に一斉に定数を加えることをバイアスといいます.

　例えば,

$$\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \circledast \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 2 & 0 & 1 \\ \hline -1 & -1 & 0 \\ \hline \end{array} + 3 = \begin{array}{|c|c|}\hline 4 & 8 \\ \hline 20 & 24 \\ \hline \end{array} + 3 = \begin{array}{|c|c|}\hline 7 & 11 \\ \hline 23 & 27 \\ \hline \end{array}$$

などとします.

パディング

入力 X の周囲に定数（ 0 など）を埋めることをパディングといいます（$\mathrm{pad} =$ 詰め物）．畳み込み演算の場合は定数は 0 を上下左右同じ幅でパディングします．パディングを行うことにより，入力と出力の型を揃えたりすることができるようになります．

例えば周囲に 1 列 0 をパディングすると

$$\begin{array}{|c|c|c|c|c|c|}\hline 0&0&0&0&0&0\\\hline 0&1&2&3&4&0\\\hline 0&5&6&7&8&0\\\hline 0&9&10&11&12&0\\\hline 0&13&14&15&16&0\\\hline 0&0&0&0&0&0\\\hline\end{array} \circledast \begin{array}{|c|c|c|}\hline 1&1&1\\\hline 2&0&1\\\hline -1&-1&0\\\hline\end{array} = \begin{array}{|c|c|c|c|}\hline -3&-6&-5&-9\\\hline 0&4&8&-2\\\hline 8&20&24&6\\\hline 33&71&77&53\\\hline\end{array}$$

などとなります．

ストライド

畳み込み演算の際，これまではフィルターを 1 マスずつずらして来たましたが，このずらすマスの数を 2 以上の整数に変えることができます．この整数をストライドといいます．

例えば，

$$Y = \begin{array}{|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&1\\\hline 0&1&2&3&4&0&1\\\hline 0&5&6&7&8&0&1\\\hline 0&9&10&11&12&0&1\\\hline 0&13&14&15&16&0&1\\\hline 0&0&0&0&0&0&1\\\hline 1&1&1&1&1&1&1\\\hline\end{array} \circledast \begin{array}{|c|c|c|}\hline 1&1&1\\\hline 2&0&1\\\hline -1&-1&0\\\hline\end{array}$$

においてストライドが 2 のとき，2 マスずつずれてフィルターを適用するので出力のサイズは 3×3 となり，

$$Y = \begin{array}{|c|c|c|}\hline -3&-5&2\\\hline 8&24&18\\\hline 25&43&16\\\hline\end{array}$$

などととなります．

実際に計算してみましょう

公式

パディングサイズを P，ストライドを S とするとき，出力サイズは $\mathrm{OH} \times \mathrm{OW}$，ただし

$$\mathrm{OH} = \frac{\mathrm{H} + 2\mathrm{P} - \mathrm{FH}}{\mathrm{S}} + 1$$
$$\mathrm{OW} = \frac{\mathrm{W} + 2\mathrm{P} - \mathrm{FW}}{\mathrm{S}} + 1$$

となります．ただし，割り算において割り切れない場合は小数部分を切り捨てるものとします．
これを Python 風に書くと

$$\mathrm{OH} = (\mathrm{H} + 2 * \mathrm{P} - \mathrm{FH})//\mathrm{S} + 1$$
$$\mathrm{OW} = (\mathrm{W} + 2 * \mathrm{P} - \mathrm{FW})//\mathrm{S} + 1$$

となります．

問題：上の式を示してください．

1.3　畳み込み演算（チャンネル，ミニバッチあり）

　これまで見てきた畳み込み演算は縦方向と横方向からなる 2 階の配列を入力とする演算でした．しかし，実際にはこれにチャンネル方向とミニバッチのデータ番号方向を加えた 4 階の配列を扱う必要があります．ここでチャンネルというのは，カラー画像における RGB (Red, Green, Blue) のように，データの多様性を保つためのものです．

入出力データ配列は 4 つの次元を持つ

N：ミニバッチのサイズ
C：入力のチャンネル数　　OC：出力のチャンネル数
H：入力データの高さ　　FH：フィルターの高さ
W：入力データの幅　　FW：フィルターの幅

とするとき，畳み込み層への入力 X，フィルター F，出力 Y はそれぞれ

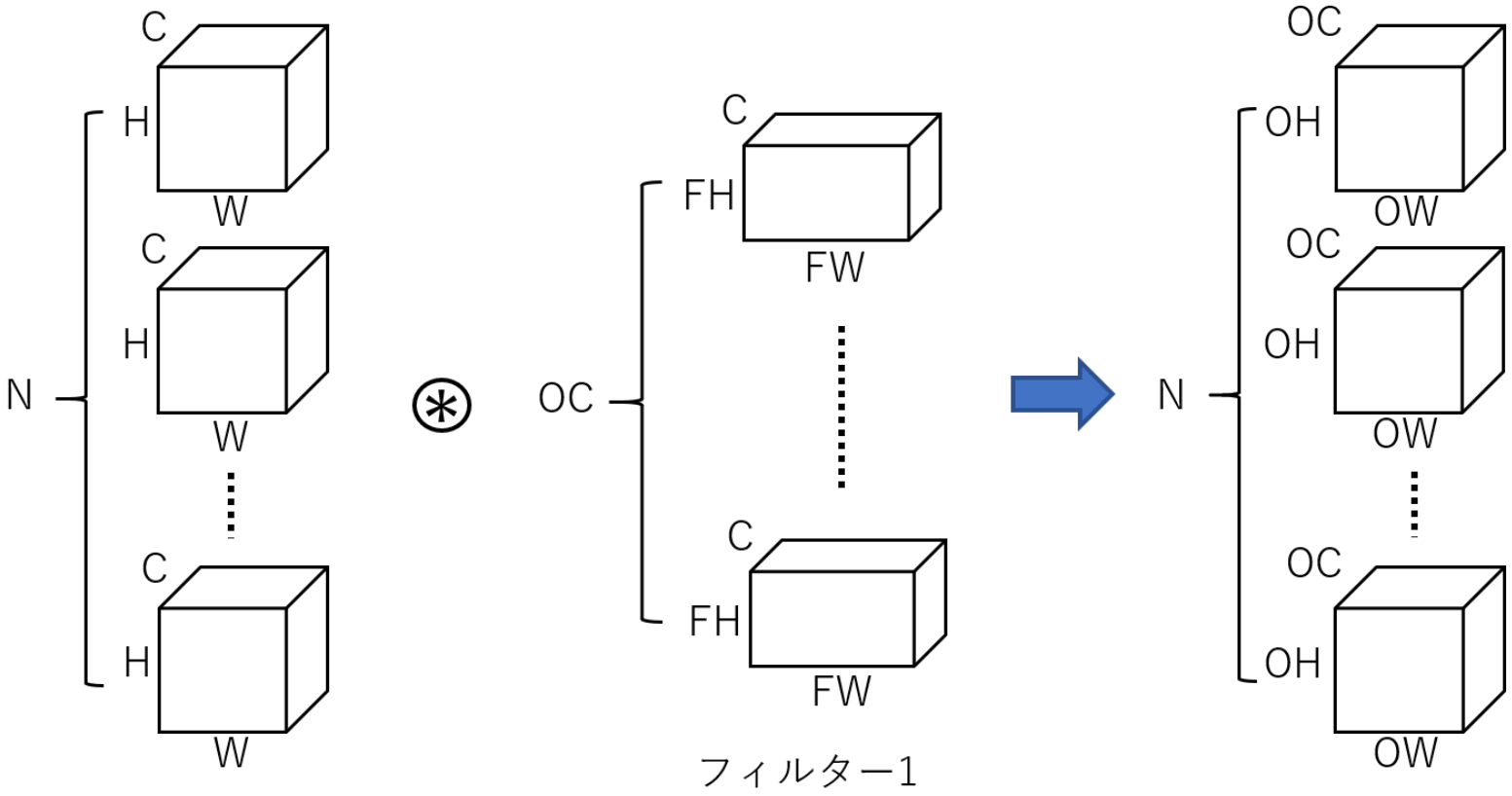
$$
\begin{aligned}
X &: (\mathrm{N}, \mathrm{C}, \mathrm{H}, \mathrm{W}) \\
F &: (\mathrm{OC}, \mathrm{C}, \mathrm{FH}, \mathrm{FW}) \\
Y &: (\mathrm{N}, \mathrm{OC}, \mathrm{OH}, \mathrm{OW})
\end{aligned}
$$

型の配列となります．ただし，出力の高さ，幅は，

$$
\begin{aligned}
\mathrm{OH} &= \frac{H + 2\mathrm{P} - \mathrm{FH}}{\mathrm{S}} + 1 \\
\mathrm{OW} &= \frac{W + 2\mathrm{P} - \mathrm{FW}}{\mathrm{S}} + 1
\end{aligned}
$$

（P：パディング，S：ストライド）で与えられます．

畳み込み演算の次元の図



im2col

畳み込み演算を実装しようとするとネックになるのが入力 X からフィルターと同じ形のベクトルを必要な数だけ取り出す部分になります.

例えば

入力 $X =$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

から $\tilde{X} =$

1	2	3	2	3	4
5	6	7	6	7	8
9	10	11	10	11	12
5	6	7	6	7	8
9	10	11	10	11	12
13	14	15	14	15	16

という $2 \times 2 \times 3 \times 3$ 型配列を取り出す必要があります.

単純に実装すると結果の $\mathrm{OW} \times \mathrm{OH} = 2 \times 2$ の部分について for 文を用いて以下のように書けます.

```
xt = np.zeros((OH, OW, FH, FW))
for oh in range(OH):
  for ow in range(OW):
    xt[oh, ow, :, :] = x[oh:oh + FH, ow:ow + FW]
```

im2col（つづき）

しかし，この方法では for 文を $\mathrm{OH} \times \mathrm{OW}$ 回計算しなければならず，X が大きくなったとき，処理に時間がかかってしまいます．そこで，X から出力データの大きさである 2×2 の大きさの部分をフィルターのサイズ分だけずらす操作により，まず，

$$\hat{X} = \begin{array}{|c|c|c|} \hline \boxed{1}\ 2 \\ 5\ \ 6 & \boxed{2}\ 3 \\ 6\ \ 7 & \boxed{3}\ 4 \\ 7\ \ 8 \\ \hline \boxed{5}\ 6 \\ 9\ \ 10 & \boxed{6}\ 7 \\ 10\ \ 11 & \boxed{7}\ 8 \\ 11\ \ 12 \\ \hline \boxed{9}\ 10 \\ 13\ \ 14 & \boxed{10}\ 11 \\ 14\ \ 15 & \boxed{11}\ 12 \\ 15\ \ 16 \\ \hline \end{array}$$

という $3 \times 3 \times 2 \times 2$ 型の配列を作り，次に，四角で囲まれた数字を並べることにより，$2 \times 2 \times 3 \times 3$ 型配列の $(0, 0)$ 成分が取り出すという方法を考えます．

im2col（つづき）

2 つ目の操作は，$\hat{X}$ に対して $\tilde{X}[o_h, o_w, f_h, f_w] = \hat{X}[f_h, f_w, o_h, o_w]$ と並べ直せば良いので，このアルゴリズムは以下の様に書けます．

```
xh = np.zeros((FH, FW, OH, OW))
for fh in range(FH):
  for fw in range(FW):
    xh[fh, fw, :, :] =x[fh:fh + OH, fw:fw + OW]
xt = xh.transpose(2, 3, 0, 1) #並べ替え
```

この方法では for 文によるループがフィルターサイズ $\mathrm{FH} \times \mathrm{FW}$ で済むので効率的です．このアルゴリズムを利用した実装方法は，im2col と呼ばれます．

1.4　畳み込み演算に関する演習

使用するノートブック: `3-1_畳み込み演算.ipynb`

このノートブックでは，畳み込み演算に関する演習を行います．基本的にはこれまでスライドで示してきたことを NumPy で実装し，確かめます．最後に簡単な問題があります．

以下のことに注意してノートブックを実行してください．

- im2col の実装，特に $\tilde{X}$，$\hat{X}$ の計算の仕方
- 4 階の配列の扱い
- 配列のスライス

1.5　プーリング

プーリングは，2 階の入力データからそれよりも次元の小さい 2 階のデータを得る演算です．畳み込み演算と違ってフィルタはパラメータを持ちません．

次の例は 4×4 の入力 X に対して 枠のサイズが 2×2 の マックスプーリングと呼ばれる操作を行った例です．

$$X = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 16 & 15 & 14 & 13 \\ \hline 12 & 11 & 10 & 9 \\ \hline \end{array} \rightarrow Y = \begin{array}{|c|c|} \hline 6 & 8 \\ \hline 16 & 14 \\ \hline \end{array}$$

出力 Y の $(0,0)$ 成分 ’6’ は X の $\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}$ の部分の最大値です．同様に $(0,1)$ 成分 ’8’ は X の $\begin{array}{|c|c|} \hline 3 & 4 \\ \hline 7 & 8 \\ \hline \end{array}$ の部分の最大値です． $(1,0)$ 成分，$(1,1)$ 成分についても同様です．

プーリング（つづき）

このように，畳み込み演算ではフィルター F との内積をとっていた部分がプーリングでは「最大値をとる」という操作に置き換わっています．また，最大値 をとる X の部分が重なり合わないように，枠のサイズとストライドの幅を等しくとります．したがって枠は常に正方形です（ストライドも縦横で変えれば長方形のフィルターを考えられないことはないですが，あまり使われません）．

実際の計算では畳み込み演算と同様に $\tilde{X}$ を経由させます．

$$X = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 16 & 15 & 14 & 13 \\ \hline 12 & 11 & 10 & 9 \\ \hline \end{array} \rightarrow \tilde{X} = \begin{array}{|cc|cc|} \hline 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ \hline \end{array} \rightarrow Y = \begin{array}{|c|c|} \hline 6 & 8 \\ \hline 14 & 16 \\ \hline \end{array}$$

チャンネル・ミニバッチ対応

プーリングはチャンネルごとに施します． したがって，層の前後でチャンネル数は変わりません．もちろんミニバッチのサイズも変わりません．それ以外は畳み込み層とほとんど同じです．

注：プーリングはパラメータを持ちません．枠の大きさ $S \times S$ のマックスプーリングと，フィルターの高さ，幅，ストライドがともに S の畳み込みで，入出力のチャンネル数が変わらないものとでは，出力サイズは同じですが，学習可能なフィルターを使っているかどうか，入力チャンネルについて和をとるかどうかという点で異なります（畳み込みの場合はフィルターが学習可能で入力チャンネルについて和をとります）．

1.6　プーリング演算に関する演習

使用するノートブック: 3-2_プーリング演算.ipynb

このノートブックでは，（マックス）プーリング演算に関する演習を行います．畳み込み演算のときと同様にこれまでスライドで示してきたことを NumPy で実装し，確かめます．最後に簡単な問題があります．

以下のことに注意してノートブックを実行してください．

- im2col の実装，特に $\tilde{X}$，$\hat{X}$ の計算の仕方
- 4 階の配列の扱い
- 配列のスライス

2　畳み込みニューラルネットワークの実装

CNN を実現するためには，畳み込み層における誤差逆伝播をクラスとして実装する必要があります.

ただし，畳み込み層の逆伝播はかなり複雑です.

2.1　畳み込み層

誤差逆伝播 ♯

入力 X ⊛ フィルター $F =$

x_{00}	x_{01}	x_{02}	x_{03}
x_{10}	x_{11}	x_{12}	x_{13}
x_{20}	x_{21}	x_{22}	x_{23}
x_{30}	x_{31}	x_{32}	x_{33}

$\circledast$

f_{00}	f_{01}
f_{10}	f_{11}

$=$

y_{00}	y_{01}	y_{02}
y_{10}	y_{11}	y_{12}
y_{20}	y_{21}	y_{22}

$= Y$

誤差 $E = E(Y)$ という例を考えてみましょう．x_{11} は

$Y =$

$x_{00}f_{00} + x_{01}f_{01}$ $+x_{10}f_{10} + x_{11}f_{11}$	$x_{01}f_{00} + x_{02}f_{01}$ $+x_{11}f_{10} + x_{12}f_{11}$	$x_{02}f_{00} + x_{03}f_{01}$ $+x_{12}f_{10} + x_{13}f_{11}$
$x_{10}f_{00} + x_{11}f_{01}$ $+x_{20}f_{10} + x_{21}f_{11}$	$x_{11}f_{00} + x_{12}f_{01}$ $+x_{21}f_{10} + x_{22}f_{11}$	$x_{12}f_{00} + x_{13}f_{01}$ $+x_{22}f_{10} + x_{23}f_{11}$
$x_{20}f_{00} + x_{21}f_{01}$ $+x_{30}f_{10} + x_{31}f_{11}$	$x_{21}f_{00} + x_{22}f_{01}$ $+x_{31}f_{10} + x_{32}f_{11}$	$x_{22}f_{00} + x_{23}f_{01}$ $+x_{32}f_{10} + x_{33}f_{11}$

の 4 つの成分に現れるので,

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial y_{00}} f_{11} + \frac{\partial E}{\partial y_{01}} f_{10} + \frac{\partial E}{\partial y_{10}} f_{01} + \frac{\partial E}{\partial y_{11}} f_{00}$$

と計算できます.

誤差逆伝播（つづき）記号の節約のために $\dfrac{\partial E}{\partial y_{ij}} = dy_{ij}$，$\dfrac{\partial E}{\partial x_{ij}} = dx_{ij}$ などと書くと，上と同様に

$$\frac{\partial E}{\partial X} = \begin{array}{|c|c|c|c|}\hline dy_{00}f_{00} & dy_{00}f_{01}+dy_{01}f_{00} & dy_{01}f_{01}+dy_{02}f_{00} & dy_{02}f_{01} \\ \hline \begin{array}{c} dy_{00}f_{10} \\ +dy_{10}f_{00} \end{array} & \begin{array}{c} dy_{00}f_{11}+dy_{01}f_{10} \\ +dy_{10}f_{01}+dy_{11}f_{00} \end{array} & \begin{array}{c} dy_{01}f_{11}+dy_{02}f_{10} \\ +dy_{11}f_{01}+dy_{12}f_{00} \end{array} & \begin{array}{c} dy_{02}f_{11} \\ +dy_{12}f_{01} \end{array} \\ \hline \begin{array}{c} dy_{10}f_{10} \\ +dy_{20}f_{00} \end{array} & \begin{array}{c} dy_{10}f_{11}+dy_{11}f_{10} \\ +dy_{20}f_{01}+dy_{21}f_{00} \end{array} & \begin{array}{c} dy_{11}f_{11}+dy_{12}f_{10} \\ +dy_{21}f_{01}+dy_{22}f_{00} \end{array} & \begin{array}{c} dy_{12}f_{11} \\ +dy_{22}f_{01} \end{array} \\ \hline dy_{20}f_{10} & dy_{20}f_{11}+dy_{21}f_{10} & dy_{21}f_{11}+dy_{22}f_{10} & dy_{22}f_{11} \\ \hline \end{array}$$

と計算できます．実装の際はこの計算をするのに途中で

$$d\tilde{X} = \begin{array}{|c|c|c|}\hline dy_{00}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{01}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{02}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} \\ \hline dy_{10}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{11}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{12}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} \\ \hline dy_{20}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{21}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} & dy_{22}\begin{pmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{pmatrix} \\ \hline \end{array}$$

を経由して効率よく計算します．

誤差逆伝播（つづき）

パラメータ F に関する微分：また，f_{00} は

$$Y = \begin{array}{|c|c|c|}\hline x_{00}f_{00}+x_{01}f_{01} & x_{01}f_{00}+x_{02}f_{01} & x_{02}f_{00}+x_{03}f_{01} \\ +x_{10}f_{10}+x_{11}f_{11} & +x_{11}f_{10}+x_{12}f_{11} & +x_{12}f_{10}+x_{13}f_{11} \\ \hline x_{10}f_{00}+x_{11}f_{01} & x_{11}f_{00}+x_{12}f_{01} & x_{12}f_{00}+x_{13}f_{01} \\ +x_{20}f_{10}+x_{21}f_{11} & +x_{21}f_{10}+x_{22}f_{11} & +x_{22}f_{10}+x_{23}f_{11} \\ \hline x_{20}f_{00}+x_{21}f_{01} & x_{21}f_{00}+x_{22}f_{01} & x_{22}f_{00}+x_{23}f_{01} \\ +x_{30}f_{10}+x_{31}f_{11} & +x_{31}f_{10}+x_{32}f_{11} & +x_{32}f_{10}+x_{33}f_{11} \\ \hline \end{array}$$

の 9 つの成分に現れるので，

$$\begin{aligned}\frac{\partial E}{\partial f_{00}} = \quad & dy_{00}x_{00}+dy_{01}x_{01}+dy_{02}x_{02} \\ & +dy_{10}x_{10}+dy_{11}x_{11}+dy_{12}x_{12} \\ & +dy_{20}x_{20}+dy_{21}x_{21}+dy_{22}x_{22}\end{aligned}$$

と計算できます．

誤差逆伝播（つづき）

同様に

$$\frac{\partial E}{\partial F} = \begin{array}{|c|c|}\hline \begin{array}{r} dy_{00}x_{00} + dy_{01}x_{01} + dy_{02}x_{02} \\ +dy_{10}x_{10} + dy_{11}x_{11} + dy_{12}x_{12} \\ +dy_{20}x_{20} + dy_{21}x_{21} + dy_{22}x_{22} \end{array} & \begin{array}{r} dy_{00}x_{01} + dy_{01}x_{02} + dy_{02}x_{03} \\ +dy_{10}x_{11} + dy_{11}x_{12} + dy_{12}x_{13} \\ +dy_{20}x_{21} + dy_{21}x_{22} + dy_{22}x_{23} \end{array} \\ \hline \begin{array}{r} dy_{00}x_{10} + dy_{01}x_{11} + dy_{02}x_{12} \\ +dy_{10}x_{20} + dy_{11}x_{21} + dy_{12}x_{22} \\ +dy_{20}x_{30} + dy_{21}x_{31} + dy_{22}x_{32} \end{array} & \begin{array}{r} dy_{00}x_{11} + dy_{01}x_{12} + dy_{02}x_{13} \\ +dy_{10}x_{21} + dy_{11}x_{22} + dy_{12}x_{23} \\ +dy_{20}x_{31} + dy_{21}x_{32} + dy_{22}x_{33} \end{array} \\ \hline \end{array}$$

と計算できます．これも，

$$dY = \begin{array}{|ccc|}\hline dy_{00} & dy_{01} & dy_{02} \\ dy_{10} & dy_{11} & dy_{12} \\ dy_{20} & dy_{21} & dy_{22} \\ \hline \end{array} \quad \text{と} \quad \hat{X} = \begin{array}{|ccc|ccc|}\hline x_{00} & x_{01} & x_{02} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{21} & x_{22} & x_{23} \\ \hline x_{10} & x_{11} & x_{12} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{31} & x_{32} & x_{33} \\ \hline \end{array}$$

を利用して効率よく計算できます．

2.2　畳み込み層に関する演習

使用するノートブック:　3-3_畳み込み層.ipynb

このノートブックでは畳み込み層を実装し，数値微分との比較テストを行います．以下のことに注意して各ノートブックを実行してください．

1. 高階配列の演算
2. im2col の使い方：im2col のアルゴリズムをコード中で用いています
3. 入力と出力の配列としての型

2.3　プーリング層

逆伝播

プーリング層の順伝搬が

$$X = \begin{array}{|cccc|}\hline x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \\ \hline \end{array} \rightarrow \tilde{X} = \begin{array}{|cc|cc|}\hline x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ \hline x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \\ \hline \end{array}$$

$$\rightarrow Y = \begin{array}{|cc|}\hline y_{00} & y_{01} \\ y_{10} & y_{11} \\ \hline \end{array} = \begin{array}{|cc|}\hline x_{11} & x_{13} \\ x_{20} & x_{22} \\ \hline \end{array} \rightarrow E$$

のとき（$\tilde{X}$ の各一ブロック内で赤字の文字が最大），誤差逆伝搬は

$$dY = \begin{array}{|cc|}\hline dy_{00} & dy_{01} \\ dy_{10} & dy_{11} \\ \hline \end{array} \rightarrow d\tilde{X} = \begin{array}{|cc|cc|}\hline 0 & 0 & 0 & 0 \\ 0 & dy_{00} & 0 & dy_{01} \\ \hline dy_{10} & 0 & dy_{11} & 0 \\ 0 & 0 & 0 & 0 \\ \hline \end{array} \rightarrow dX = \begin{array}{|cccc|}\hline 0 & 0 & 0 & 0 \\ 0 & dy_{00} & 0 & dy_{01} \\ dy_{10} & 0 & dy_{11} & 0 \\ 0 & 0 & 0 & 0 \\ \hline \end{array}$$

となります.

2.4　プーリング層に関する演習

使用するノートブック:　3-4_プーリング層.ipynb

このノートブックではプーリング層を実装し，数値微分との比較テストを行います．以下のことに注意して各ノートブックを実行してください．また，ノートブックの最後にある問題を解いてください．

1. 畳み込み層との違い
2. 配列の最大値をとる関数：numpy.max() を用います
3. 配列の最大値をとる関数の微分：numpy.argmax() を用います

2.5　畳み込みニューラルネットワークの実装に関する演習

使用するノートブック:　3-5-1_CNN.ipynb

このノートブックでは畳み込みニューラルネットワークを実装します．以下のことに注意して各ノートブックを実行してください．また，ノートブックの最後にある問題を解いてください．

1. 流れるデータ配列の型
2. 計算量（計算時間）
3. 精度

2.6　GPU の利用に関する演習

使用するノートブック:　3-5-2_CNN_GPU.ipynb

GPU

前節の演習でも分かるように CNN の計算量は非常に大きく，計算に時間がかかります．この問題を解決するために，一般的な計算で使用される CPU（Central Processsing Unit）の代わりに，配列の計算に GPU (Graphics Processing Unit) が用いられます．GPU は元々 3D グラフィックスの描画や，画像処理を高速に計算できるように設計された演算装置であり，CPU よりもはるかに高速に配列の計算を行うことができます．

特にディープラーニングでは行列計算を多く行うため，GPU を用いて高速化することができます．配列計算に GPU を利用するには，NVIDIA が開発・提供している CUDA（Compute Unified Device Architecture) というプラットフォーム／プログラミング言語を利用するのが一般的です．

もちろん TsnsorFlow や PyTorch も CUDA に対応していますが，NumPy と同様の書き方で CUDA を利用できるライブラリーに，プリファードネットワークス（日本の会社です）が提供する CuPy があります．本演習では CuPy を用います．

3　CNN の利用

本節では CNN の利用のされ方についてまとめ，その際のいくつかの工夫について学びます．

CNN はタスクに応じて出力を適切に設計することにより，画像の回帰や分類以外にも，一般物体認識や生成モデル，回帰結合型畳み込みニューラルネットワーク等様々な場面で用いられます．このようなネットワークの出力の設計上の工夫を構造出力といいます．

3.1　データの種類

　データの種類に応じてデーター一つの次元（階数）とチャンネルの有無が変わります．以下のようなものが代表的です．（バッチの方向はカウントしていません．）

	単一チャンネル	多チャンネル
1 次元	音声波形，DNA 配列	スケルトンアニメーション (骨格の動き)
2 次元	モノクロ画像 フーリエ変換された音声データ	カラー画像
3 次元	モノクロ動画 3 次元密度データ（CT スキャン）	カラー動画

なお，MNIST 手書き数字データの認識でもそうだったように，入力が単一チャンネルであっても，中間層は多チャンネルとするのが一般的です．

3.2　データ増大

オリジナルの画像：例

https://upload.wikimedia.org/wikipedia/commons/6/6f/Silk_Way_Rally_2011.jpg

を加工して学習データを増やす手法です．どれもランダムに施します．

シフト：

拡大縮小：

カット：

データ増大（つづき）

反転：

回転：

明暗：

剪断（線形変換）：

ただし，上の写真の場合，例えば上下さかさまに写っていることはほとんどないので，このような場合には上下反転をしてはいけません．データ増大はあくまで本来のデータの分布に近づけることでモデルの汎化性能を高めるために行います．

3.3　データ増大に関する演習

使用するノートブック:　`3-6_Augmentation.ipynb`

このノートブックでは Cifar10 というカラー画像のデータセットに対して, データ増大を用いて畳み込みニューラルネットワークを学習します．NumPy-CuPy による実装, Keras-TensorFlow による実装，PyTorch による実装を比較します.

1. 学習データに対する精度とテストデータに対する精度の差
2. 計算のボトルネックはどこにあるか：畳み込み演算も計算量が大きいですが，実はデータ増大も画像を一枚一枚処理する必要があるので時間がかかります．これを回避するには，あらかじめ学習とは別にデータ増大しておく，学習と並行して別の CPU などでデータ増大を行う，などの方法が考えられます.

3.4　特徴量の転移

あるタスクについて学習させた学習済みのモデルがあるとき，それと関連する別のタスクに対して学習済みのモデルを利用する手法を転移学習と呼びます．例えば，犬，猫，馬，羊の画像の分類を学習した後，牛を加えて学習しようというときに，始めから学習するのではなく，学習済みのパラメータ（の一部—この場合は途中までの層—）を初期値として学習を行うといった場合が典型的です．

- 学習に使用できるデータが少量しかないタスクに対して，データ量の多いタスクで学習したモデルを利用する
- マルチタスク学習（画像の物体認識とクラス分類等のように関連するいくつかのタスクを同時に学習する方法）もその一種
- 自己符号化器などで事前に教師なし学習をしてから，そのモデルの一部を使って教師つきデータで再学習させる方法もその一種

ワンショット学習・ゼロショット学習

転移学習の極端な例としてワンショット学習やゼロショット学習というものがあります．

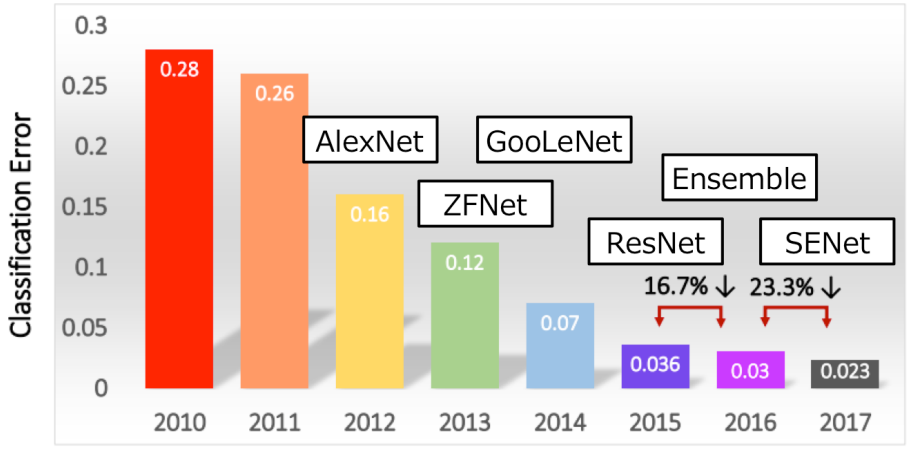
ワンショット学習　文字通り再学習において一つのデータのみを一度だけ学習させる方法．上記の例のように，犬，猫，馬，羊の画像の分類を学習した後，牛を加えて学習しようときに用いる．深層学習においてワンショット学習を行うためには，モデルがもともと特徴量表現を出力するようなモデルである必要があり，[G. Koch etal. “Siamese neural networks for one-shot image recognition” ICML Deep Learning Workshop. Vol. 2. 2015.] などが代表的．

ゼロショット学習　画像以外のテキストなどのデータを学習に加えることにより，「馬と同じくらい大きくて，少し丸くて，たてがみはない」などの情報から，牛の画像を学習させることなく，牛も分類できるようにする技術

4　CNN の発展

2012 年の画像認識コンペティション ImageNet Large Scale Visual Recognition Competition (ILSVRC) において，AlexNet と呼ばれる CNN を用いた手法が，大差をつけて優勝し，一躍深層学習が注目されて以来，画像認識では CNN が主流となり，毎年のように新たなモデルが提案されてきました．

その中でも 2015 年の ILSVRC の優勝モデルである Residual Networks (ResNet) では，離れた畳み込み層の間に「スキップ接続」を導入して非常に深いネットワークによる学習を実現しました．



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

CNN の発展 (つづき)

本節では

- VGG および GoogleNet: それぞれ 2014 年 ILSVRC の 2 位と 1 位
- Residual Networks：スキップ接続を導入して深いネットワークを実現
- Dense Networks：スキップ接続を密に施した CNN（「Dense Neural Networks 全結合ニューラルネットワーク」ではないので要注意）
- Mobile Networks：効率的な畳み込み演算をもつネットワーク

について解説します.

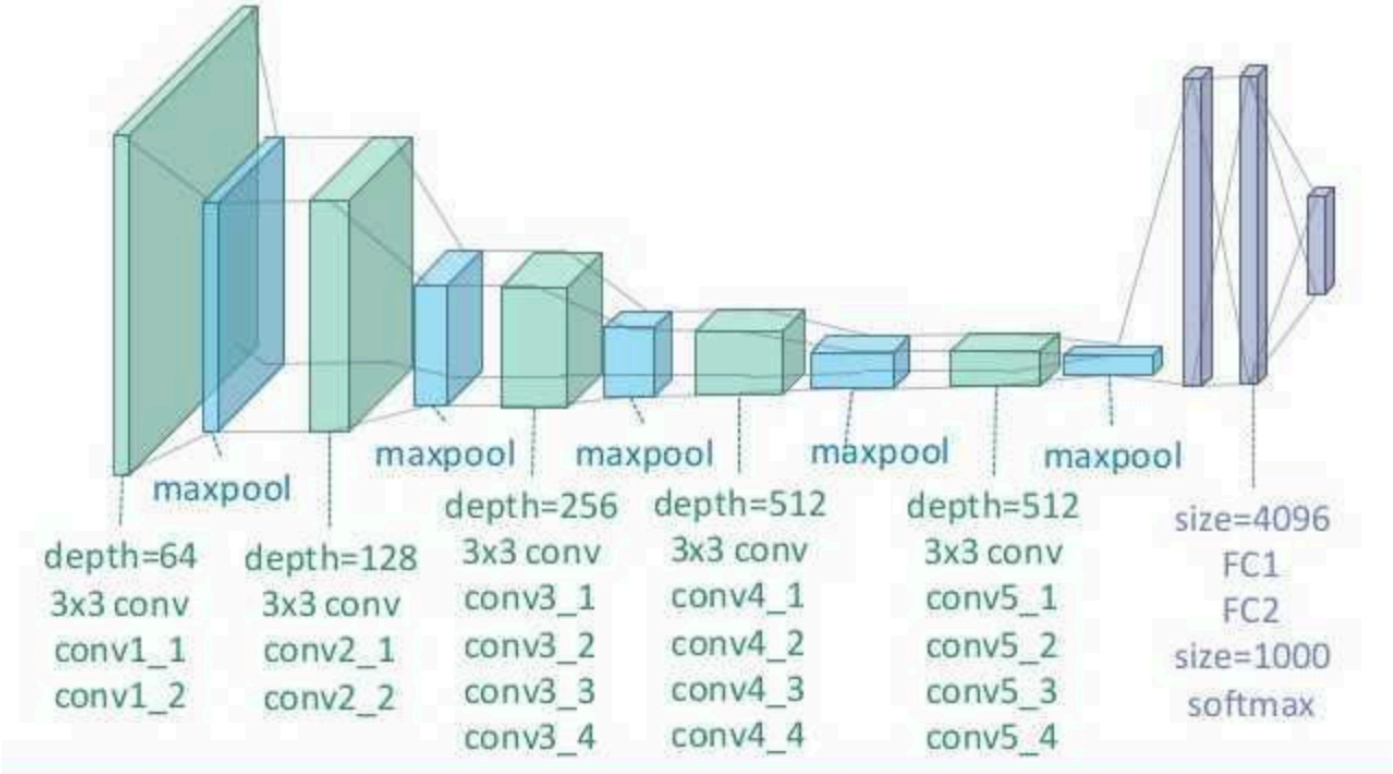
CNN は画像認識だけではなく，セグメンテーション，物体検出など様々なタスクを解くためのベースネットワークとしても広く利用されています．また，自然言語処理，音声処理，ゲーム AI 等の分野でも利用されるなど，ニューラルネットワークの中で重要な位置を占めています.

参考： Uchida, Yusuke, 畳み込みニューラルネットワークの最新研究動向 (〜2017),
`https://qiita.com/yu4u/items/7e93c454c9410c4b5427`
CNN 以外にも様々な正則化法の歴史がまとまっています.

4.1　VGG および GoogINet

VGG

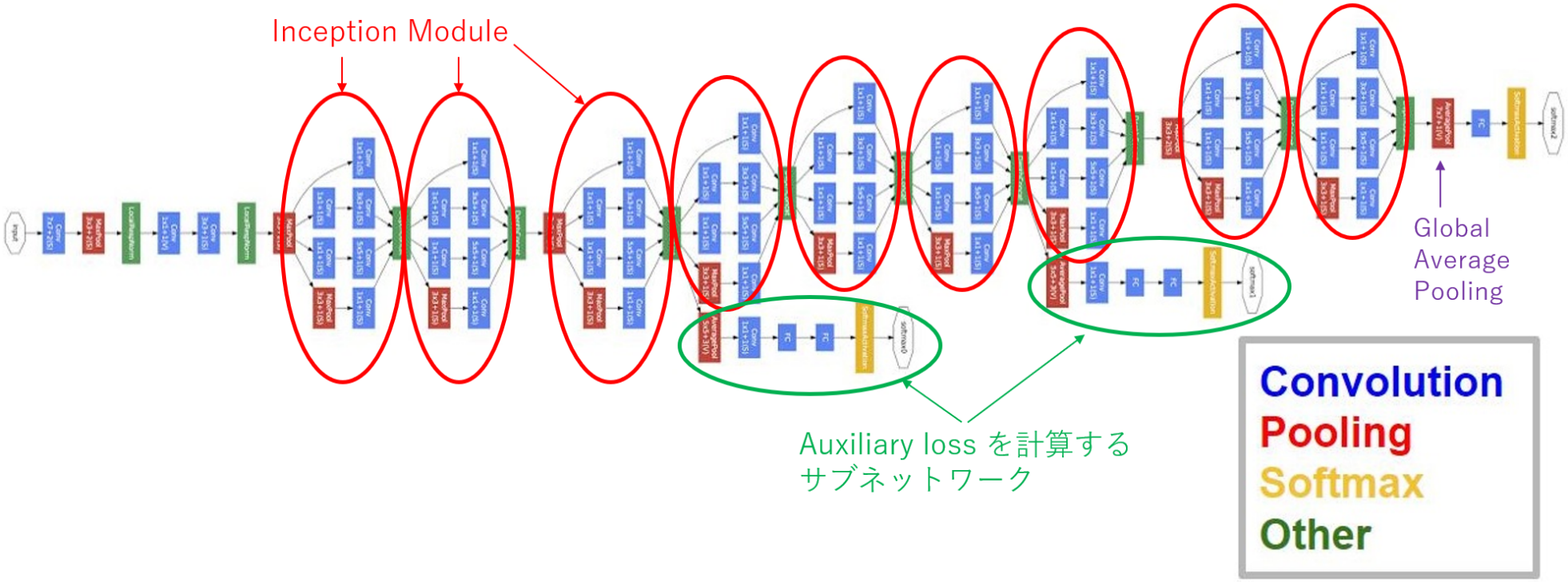
VGG[Simonyan, K., Zisserman, A., 2014] は本講座で作ってきた CNN でお手本としてきたモデルで，下図のように畳み込み層とプーリング層を積み重ねて最後に全結合層で出力します．



https://github.com/scofield7419/basic_NNs_in_frameworks/blob/master/assets/4.png

GoogLeNet

GoogLeNet[Szegedy, C. et al., 2014] は以下のように CNN を枝分かれさせながらつなげた構造を持ています．ResNet と違う点は何もしない接続は存在しないということです．



[Szegedy, C. et al., 2014]

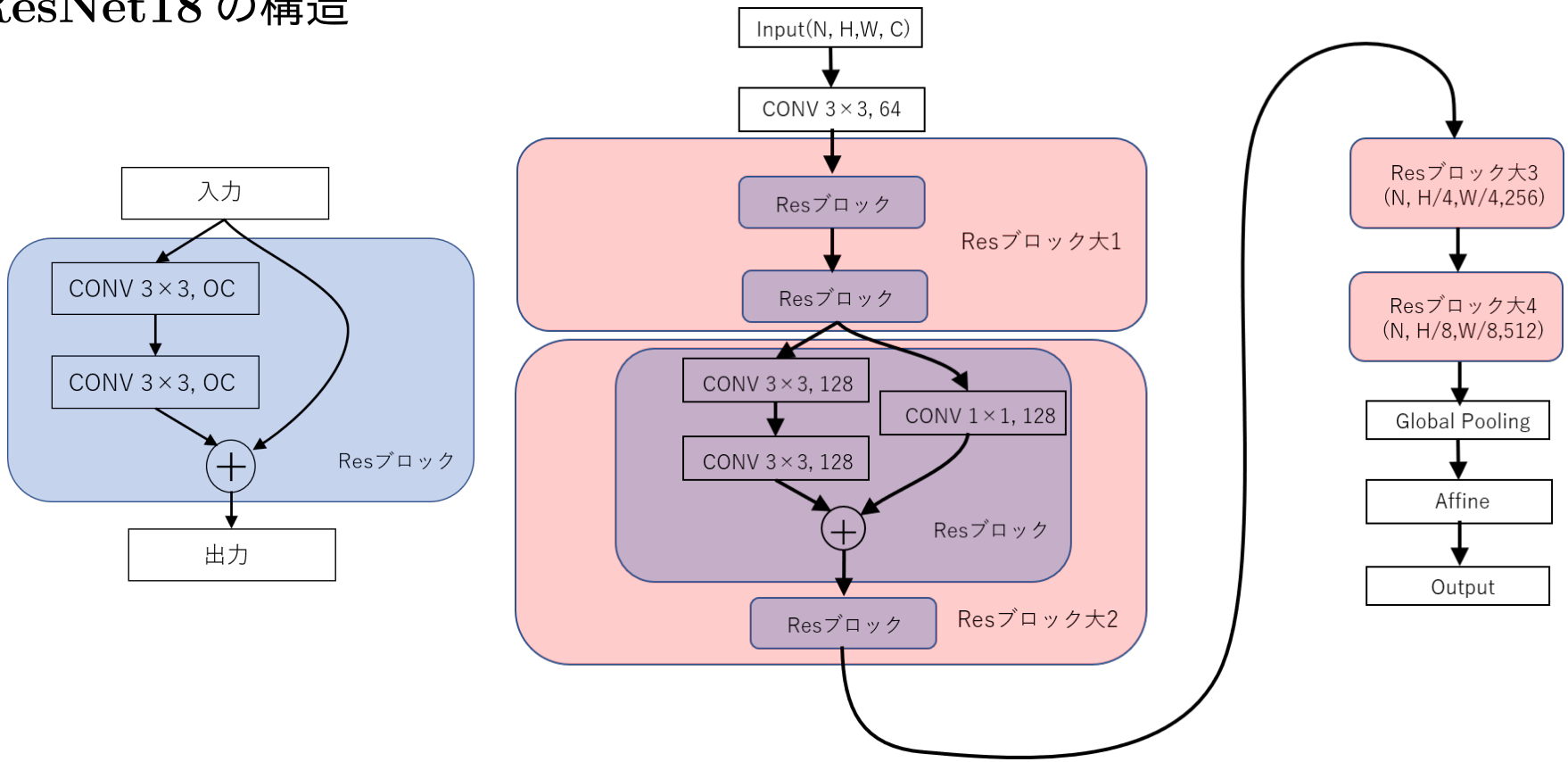
4.2 Residual Networks

全結合ネットワークや畳み込みネットワークのように層を順に伝搬して行くニューラルネットワークは，層を深くして行くと，微分などの情報が層の数に関して指数オーダーで減少または増大してしまい，伝わりにくくなってしまいます．

たとえば，層を一つ進む毎に微分が 2 倍されるとすると，10 層では $2^{10} = 1024$ 倍，20 層では $2^{20} = 1,048,576$ 倍といった具合です．その対処法として，パラメータの初期値を適切にとったり，バッチ正規化を行ったりしてきましたが，層があまりに深くなると（全結合層や畳み込み層のみ数えて 30 層以上など）情報が乱数や計算上の誤差に埋もれてしまい，やはり伝達しにくくなってしまいます．

Residual Networks (ResNet，レズネット) はこの問題に対処するために，畳み込みネットワークに，いくつかの層をスキップして情報を伝える「辺」を付け加えたニューラルネットワークで，2015 年に Kaiming He らによって提案されました [https://arxiv.org/abs/1512.03385]．

ResNet18 の構造



・左図の最小ブロックをいくつか重ねて全体を作る

・$\boxed{\text{CONV } 3 \times 3, \text{OC}}$ はサイズ 3×3 のフィルターを持ち，出力チャンネル数 OC=C の畳み込み層

・ブロック大２，３，４では，最初の Res ブロックでストライドを 2 として画像サイズを半分にする

・それ以外の畳み込み層のストライドは 1 でパディングを用いて画像の大きさを変えない

・最初の畳み込みは入力データによって変わる

・Global Average Pooling の出力サイズは (N, C)

Add

前ページの $\oplus$ は単に足し算をする層で，式で書くと入力 X，Y に対して

$$Z = X + Y$$

を出力します．X と Y は同じ型の配列でなくてはなりません．

誤差逆伝搬

$\oplus$：$Z = X + Y$ の逆伝播：

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} = \frac{\partial E}{\partial Z}$$

分岐 $X \to X_1, X_2$ での逆伝播：

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial X_1} + \frac{\partial E}{\partial X_2}$$

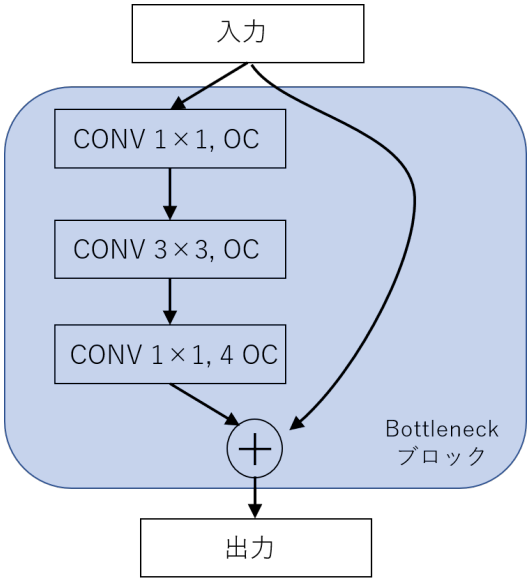
代表的な ResNet

ResNet は大きいブロックの中の（出力チャンネル数 × 小ブロック数）で分類されます.

type	ResNet18	Resnet 34	Resnet50	Resnet101	ResNet152
大ブロック 1	64 × 2	64 × 3	b64 × 3	b64 × 3	b64 × 3
大ブロック 2	128 × 2	128 × 4	b128 × 4	b128 × 4	b128 × 8
大ブロック 3	256 × 2	256 × 6	b256 × 6	b256 × 23	b256 × 36
大ブロック 4	512 × 2	512 × 3	b512 × 3	b512 × 3	b512 × 3

ここで b62，b128 などと書かれているのは “bottleneck” と呼ばれるブロックです.

ボトルネック bx, $(x = \mathrm{OC})$:



4.3 ResNet に関する演習

使用するノートブック: `3-7_ResNet.ipynb`

このノートブックでは Cifar10 というカラー画像のデータセットに対して，データ増大を用いて ResNet を学習します．実装には Keras-TensorFlow を用います．

なお，ノートブックでは Wide ResNet も実装しています．ResNet は層を深くする（積み重ねる）ことで性能を上げようというものでしたが，それ程深くないネットワークでもチャンネルを増やすことで性能を上げようという考えで考案されたのが Wide ResNet です．[Zagoruyko-Komodakis, "Wide Residual Networks", https://arxiv.org/abs/1605.07146]

1. Keras の Model() によるネットワークの生成
2. 学習済みパラメータの保存と読み込み
3. モデルの可視化
4. 学習にどれくらい時間がかかるか

5　生成モデル

　本節では生成モデルというニューラルネットワークについて学びます．生成モデルは画像や音声などの学習したデータを真似して新しいデータを生成するモデルです．

5.1　生成モデルと識別モデル

生成モデル　訓練データを学習し，それらのデータと類似した新しいデータを生成するモデルを生成モデルといいます．学習データの分布と生成データの分布（データの分布とは一つ一つのデータの発生する確率のこと）が一致するように学習させます．多くの生成モデルはエンコーダ（符号化器）とデコーダ（復号化器）からなり，符号化により潜在表現を得て，復号化により新たな（あるいは入力と同じ）データを生成します．

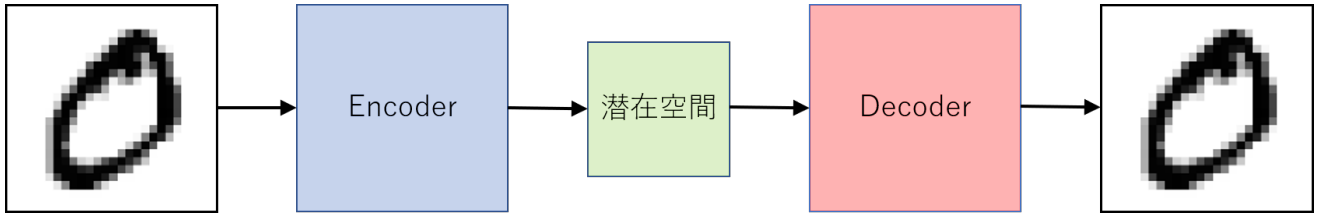
識別モデル　与えられたデータが訓練データであるか，モデルが生成したデータであるかを判別するモデルを識別モデルといいます．

5.2　オートエンコーダ

オートエンコーダはエンコーダ（符号化器）とデコーダ（復号器）という 2 つのニューラルネットワークを連結したニューラルネットワークであり，エンコーダによる次元削減を行うことにより，対象データの特徴を把握したり，中間層（潜在空間）にノイズを加えることにより，新たなデータを生成することができます．

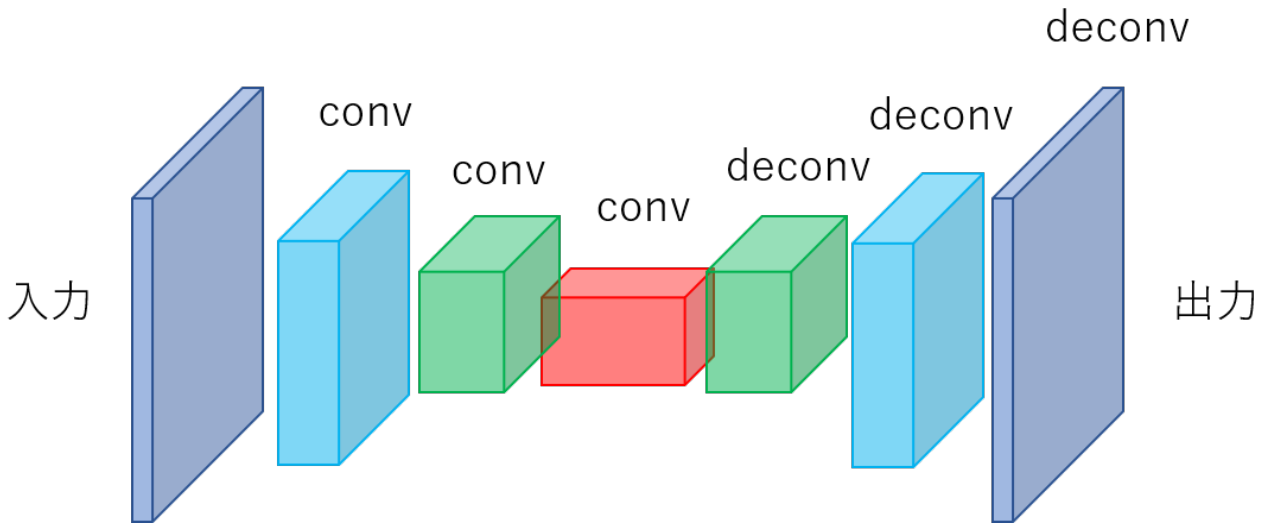
オートエンコーダの損失関数は通常，入力データと出力データの間の 2 乗誤差の和を用います．これにより，入力データと出力データが同じ分布に従うように学習されます．入力データが実数ベクトルで範囲に制約がない場合，出力層の活性化関数は恒等写像が選ばれることが多く，画像データのように値に範囲がある場合は，シグモイド関数などを用います．

オートエンコーダは異常値の検出にも用いられますが，この場合，入力データと出力データの差が大きいときに異常値と考えます．



5.3　逆畳み込み（転置畳み込み）と畳み込みオートエンコーダ

オートエンコーダは元々はアファイン層（全結合層）を積み重ねたものでしたが，現在では畳み込み演算を用いたものが一般的です．ただし問題となるのは，畳み込み演算をほどこすと画像サイズは同じか小さくなってしまい，大きくすることができないという点です．潜在表現を得るというオートエンコーダの目的に照らすと一度画像サイズを小さくする必要があります（チャンネル方向には増やしても構わない）．そこで用いられるのが「逆畳み込み」または「転置畳み込み」（この 2 つは同じもの）と呼ばれる演算です．



畳み込みオートエンコーダ

逆畳み込み演算

入力 $X = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$ にフィルター $F = \begin{array}{|c|c|}\hline 1 & -1 \\ \hline 2 & 1 \\ \hline \end{array}$ でストライド 2 の逆畳み込みを適用すると，まず X を

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 3 & 0 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

と拡張しあたとで F との畳み込み演算を行い，

$$\begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 3 & 0 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \circledast \begin{array}{|c|c|}\hline 1 & -1 \\ \hline 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 2 & 4 \\ \hline -1 & 1 & -2 & 2 \\ \hline 3 & 6 & 4 & 8 \\ \hline -3 & 3 & -4 & 4 \\ \hline \end{array} = Y$$

と計算します．

ここでストライドとは，拡張のときどれだけ X の要素を離すかというパラメータになります．

逆畳み込み演算（つづき）

ストライドが 1 のときは

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 0 \\ \hline 0 & 3 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \circledast \begin{array}{|c|c|}\hline 1 & -1 \\ \hline 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 1 & 4 & 4 \\ \hline 2 & 1 & 10 \\ \hline -3 & -1 & 4 \\ \hline \end{array} = Y$$

などとします．その際，周りにどれだけパディングするかは目的の大きさに応じて決めます．

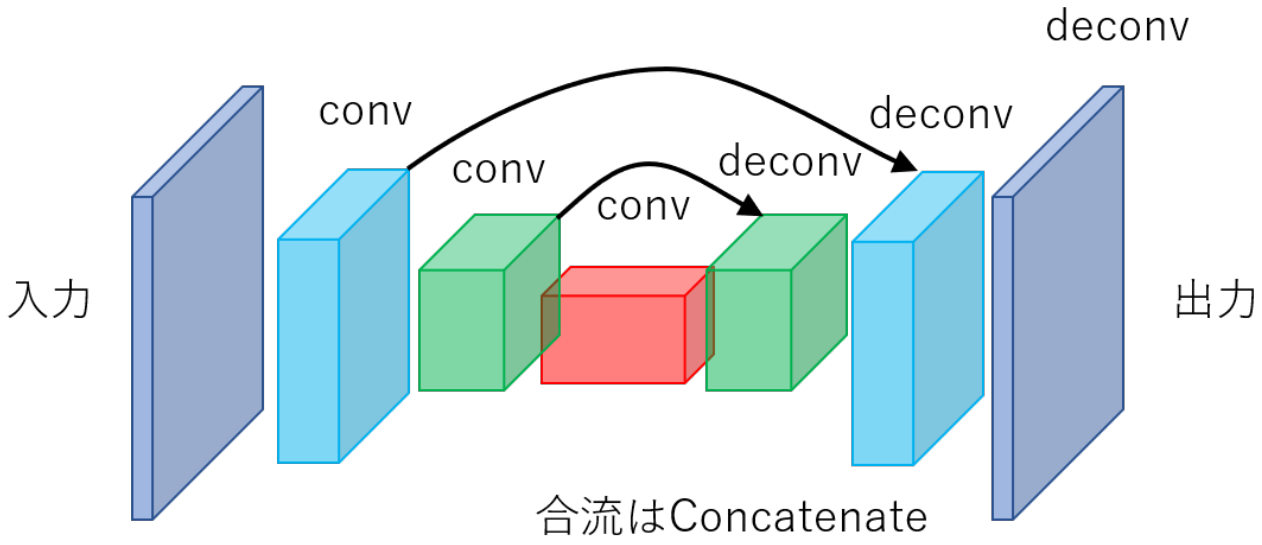
なお，逆畳み込みは Y を畳み込むと X に戻るわけではないので，畳み込みの逆演算ではありません．むしろ，畳み込みを行列の掛け算として書いたときに転置行列を逆から書けることに対応しているので転置畳み込みといった方が正確です．

また，逆畳み込みにおけるパディング数とは，畳み込みで Y から X（と同じサイズの別の行列）に行くときに必要なパディングの数のことなので注意してください．例えば上で挙げた２つの例はどちらもパディング数 0 です．こちらのサイトに分かり易いアニメーションがあります．

https://github.com/vdumoulin/conv_arithmetic

5.4　U-Net

畳み込みオートエンコーダを画像のスタイルを変換するタスクに適用することができます．例えば画像の画素一つ一つについてクラスを分類するセマンティック・セグメンテーションと呼ばれるタスクなどです．そのようなときに有効なのが畳み込みオートエンコーダにスキップ接続を付け加えた U-Net と呼ばれるモデルです．FCN（Fully Convolutional Networks）というモデルも似たような構造を持ちますが，U-Net の方が単純な構造です．



5.5　オートエンコーダ（U-Net）によるセグメンテーションに関する演習

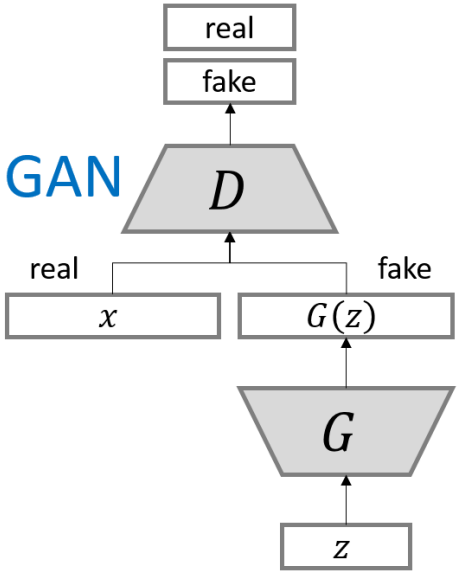
使用するノートブック:　3-8_U-Net によるセグメンテーション.ipynb

このノートブックではスキップ接続を持つ畳み込みオートエンコーダである U-Net を Keras を用いて実装し，それを利用してセマンティック・セグメンテーションというタスクに取り組みます．セマンティック・セグメンテーションというのは画像の画素一つ一つに対してそれが属するクラスを推測するというタスクです．

- Keras の転置畳み込み層のクラスは Conv2DTranspose
- U-Net の各層において縦横の大きさとチャンネル数はどのように変わるでしょうか
- セマンティック・セグメンテーションにおける誤差関数：交差エントロピーが用いられます
- データセット：心臓の MRI 画像から左心室を特定します
- 他にどのような応用が考えられるでしょうか

5.6 GAN

Generative Adversarial Networks (GAN, 敵対的生成ネットワーク) [Goodfellow et al., 2014] は生成器 Generator と，識別器 Discriminator を交互に学習させるモデルです． 生成器は乱数により発生させたノイズ $\mathbf{z}$ を入力とし，新しいデータ $G(\mathbf{z})$ を生成します．識別器は，ランダムに選ばれた訓練データ $\mathbf{x}$ または生成器の生成したデータ $\mathbf{x} = G(\mathbf{z})$ のどちらかを入力とし，それが訓練データである予測確率 $D(\mathbf{x})$ を出力します．つまり，$D(\mathbf{x}) > 0.5$ のとき $\mathbf{x}$ は訓練データであったと予測し，$D(\mathbf{x}) \leq 0.5$ のとき $\mathbf{x}$ は生成されたデータであったと予測します．



https://github.com/hwalsuklee/tensorflow-generative-model-collections

GAN の学習

GAN では，

1. 生成器のパラメータを固定して識別器のみ学習させる
2. 識別器のパラメータを固定して生成器のみ学習させる

を交互に繰り返すことで，生成器は限りなく訓練データのものに近い分布でデータを生成し，識別器はそのようなデータに対して識別の精度が高くなるように学習されます．

このような GAN の学習は絵画の「贋作者」と「鑑定家」が互いに競い合って技術を高めていく構造と似ています．

完全に学習が進み，学習データの分布 $P_{\mathrm{data}}(\mathbf{x})$ と生成データの分布 $P_{\mathrm{gen}(\mathbf{x})}$ が一致するとき，識別は不可能なので，$D(\mathbf{x})$ は 0.5 に限りなく近くなります．

注：ここでいうデータの分布は，例えば画像のときなら，ある画像一つ一つが出現する確率分布のことです．例えば犬の画像といってもたくさんありますが，「犬」の画像といったときにそれらの一つ一つが出現する確率のことです．

GAN の価値関数

識別器に対する正解ラベル y を，入力が訓練データのとき $y=1$，生成されたデータのとき $y=0$ とすると，識別器の条件付対数尤度（交差エントロピー $\times(-1)$）は

$$\log L(D(\mathbf{x}) \mid y) = y \log D(\mathbf{x}) + (1-y)\log(1-D(\mathbf{x}))$$

となります．従って，訓練データを入力するときのこの期待値は，（$y=1$ なので）

$$E_{\mathbf{x}\sim P_{\mathrm{data}}(\mathbf{x})}[\log D(\mathbf{x})] = \sum_{\mathbf{x}} P_{\mathrm{data}}(\mathbf{x}) \log D(\mathbf{x})$$

であり，生成されたデータを入力するときの期待値は，（$y=0$ なので）

$$E_{\mathbf{x}\sim P_{\mathrm{gen}}(\mathbf{x})}[\log(1-D(\mathbf{x})] = \sum_{\mathbf{x}} P_{\mathrm{gen}(\mathbf{x})} \log(1-D(\mathbf{x}))$$

となります．ここで，生成されるデータが $\mathbf{x}=G(\mathbf{z})$ となる確率はノイズが $\mathbf{z}$ となる確率 $P_{\mathrm{noise}}(\mathbf{z})$ と一致するので，

$$E_{\mathbf{x}\sim P_{\mathrm{gen}}(\mathbf{x})}[\log(1-D(\mathbf{x}))] = E_{\mathbf{z}\sim P_{\mathrm{noise}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))] = \sum_{\mathbf{z}} P_{\mathrm{noise}}(\mathbf{z}) \log(1-D(G(\mathbf{z})))$$

と書けます（実際は連続分布ですが，簡単のため離散分布として説明しました）．

GAN の価値関数（つづき）

特に訓練データを入力する割合と，生成器によるデータを入力する割合が等しいとすると（GAN ではこのようにとる），識別器の条件付対数尤度の 2 倍は，

$$V(D,G) = E_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + E_{\mathbf{z}\sim P_{\text{noise}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \quad \cdots ①$$

となります．$V(D,G)$ を GAN の価値関数といいます．

識別器はこの $V(D,G)$ を最大化するように学習されます．従って，識別器の学習は $\max_D V(D,G)$ と表せます．一方，生成器はなるべく識別器をだますように，つまり $V(D,G)$ を最小化するように学習されます．従って，生成器の学習は $\min_G V(D,G)$ と表せます．よって GAN の学習は

$$\min_G \max_D V(D,G)$$

と表せます．①の第一項はノイズ $\mathbf{z}$ および生成器 G によらないので，生成器の学習は

$$\min_G E_{\mathbf{z}\sim P_{\text{noise}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

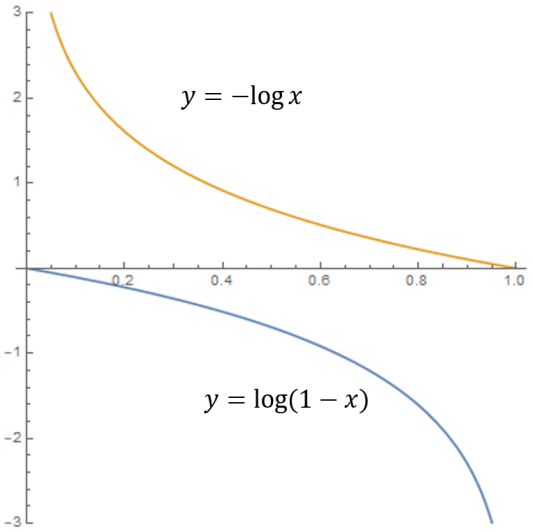
とも表せます．

GAN の価値関数（つづき）

なお，$D(G(\mathbf{z}))$ が 0 に近づき，G についての勾配が 0 に近くなって学習が進まなくなることへの対策として，生成器の学習を

$$\min_G E_{\mathbf{z}\sim P_{\text{noise}}(\mathbf{z})}[-\log D(G(\mathbf{z}))] \quad \cdots ②$$

と変更することが多いですが，このコスト関数の変更は理論的というよりは，実験的に求められたものです [Goodfellow, §20.10.4].



$y = \log(1-x)$ と $y = -\log x$ はどちらも単調減少関数

5.7 DCGAN

GAN の手法をもとに，生成器と識別器に畳み込みニューラルネットワークを用いたモデルを Deep Convolutional GAN (DCGAN) といいます．DCGAN では

- プーリング層の代わりに畳み込み層を用いる
- 生成器の出力に tanh を用いる
- 識別器で

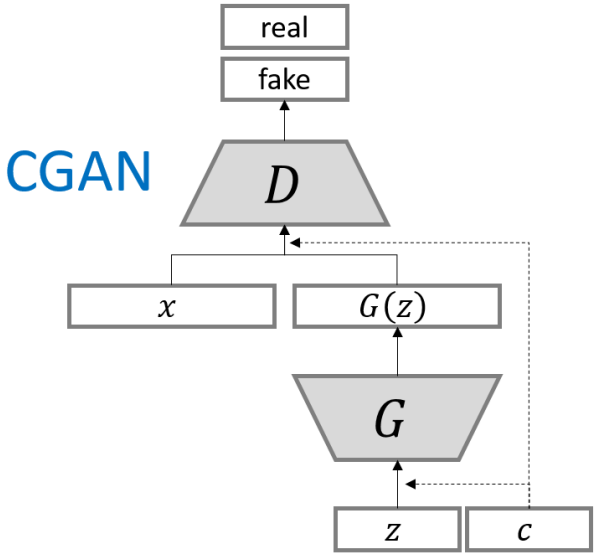
$$f(x) = \begin{cases} x & (x > 0) \\ 0.01x & (x \leq 0) \end{cases}$$

と定義される Leaky ReLU 関数を用いる

などの工夫が施されています．

5.8 Conditionnal GAN

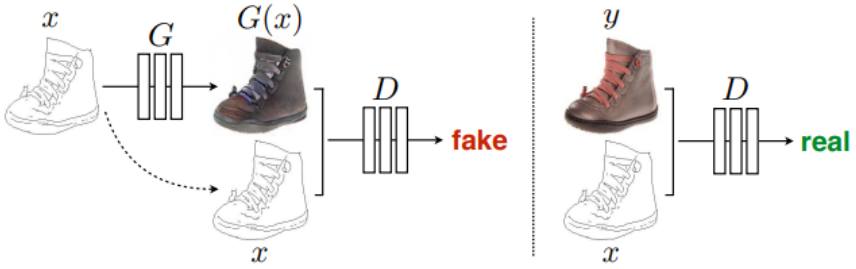
GAN は教師データの分布を真似るだけなので，訓練データにいくつかのクラスがある場合でも，どのクラスのデータを生成するのかコントロールできません．Conditional GAN では，生成器と識別器両方に同じ条件 c を与えることで識別するデータが条件 c(例えばクラス名) の教師データと似ているときのみ，識別器が本物と判断する，つまり，$D(G(\mathbf{z}, c), c) > 0.5$ とするように学習させます．



https://github.com/hwalsuklee/tensorflow-generative-model-collections

5.9　pix2pix

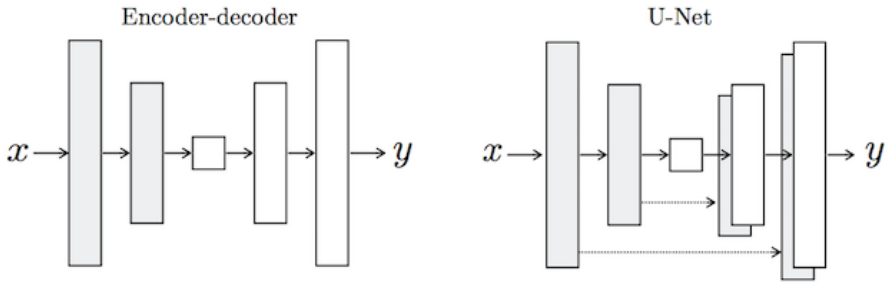
pix2pix は敵対的生成ネットワーク（GAN）を用いて入力画像に対してスタイル変換を行うニューラルネットワークモデルです [Isola, P. et al., 2017]．pix2pix は，線画の自動着色，絵画のスタイル変換（ゴッホ風の絵に変換するなど），グレースケール画像から RGB 画像化，航空写真の地図画像化などに応用されていて，ニュース等で取り上げられることも多いのでご存知の方も多いかもしれません．



[Isola, P. et al., 2017] 線画を着色しようとする場合，生成器は入力された線画に対して着色を試みます．識別器では，線画と着色画像のセットが入力されますが．着色画像は生成器で生成されたものまたは正解画像のどちらかが入力されます．識別器はどちらが入力されたか識別するように訓練されます．

生成器の構造

pix2pix の Generator のネットワークには U-Net と呼ばれるネットワークモデルが用いられています．U-Net は畳み込み層，逆畳み込み層，スキップ接続を用いた図のようなネットワークです．スキップ接続は，入出力画像間のエッジ位置の共有に効果があります．なお，pix2pix では U-Net によって次元削減を行うので，通常の GAN とは異なり，生成にノイズは用いません．また，生成器の学習では，GAN の誤差関数に，正解画像と生成画像の間の L1 誤差を加えたものを使用します．



Encoder-decoder モデルと U-Net [Isola et all, 2017]

オリジナルの提案者らによる実装例 https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

5.10　pix2pix に関する演習

使用するノートブック: `6-2_pix2pix-Keras.ipynb`

このノートブックでは keras を用いて pix2pix を実装したライブラリーを利用して白黒画像の着彩というタスクに取り組みましょう．本ノートブックはオリジナルの提案者の一人である Phillip Isola 氏によるコードを TensorFlow2 用に書き換えたものです．

- ともかくディープラーニングの華ともいえる pix2pix を使ってみましょう
- できたら他のスタイル変換タスクにも挑戦してみてください

6　一般物体検出—画像の局在化・検知・セグメンテーション

　与えられた画像の中から，定められた物体（人，車，製品など）の位置とクラスなどを検出するタスクを一般物体検出といいます．その中でも以下のタスクがよく用いられます．

分類　画像に写っている物体のクラスを特定する

画像の局在化（位置特定）　対象となる物体を囲む矩形を特定する

検知　対象となる物体があれば，そのクラスと矩形位置を特定する

セマンティック・セグメンテーション　画像中の各画素がどの物体のクラスに属するか割り当てることにより，物体の形を把握する

Faster R-CNN
https://arxiv.org/abs/1506.01497

6.1 Faster R-CNN

R-CNN

一般物体検出のアルゴリズムに R-CNN (Regions with CNN features) というアルゴリズムがあります．R-CNN のアルゴリズムは以下のような流れです．

1. 物体候補領域（矩形位置の候補）を Selective Search という既存手法を用いて求める（2000 個程度）
2. 物体候補の領域画像を全て一定の大きさにリサイズして CNN にかけて特徴量を取り出す
3. 取り出した特徴量を使って複数の SVM（サポートベクトルマシン） によってカテゴリ識別を行い，回帰によってより正確な矩形位置を推定する

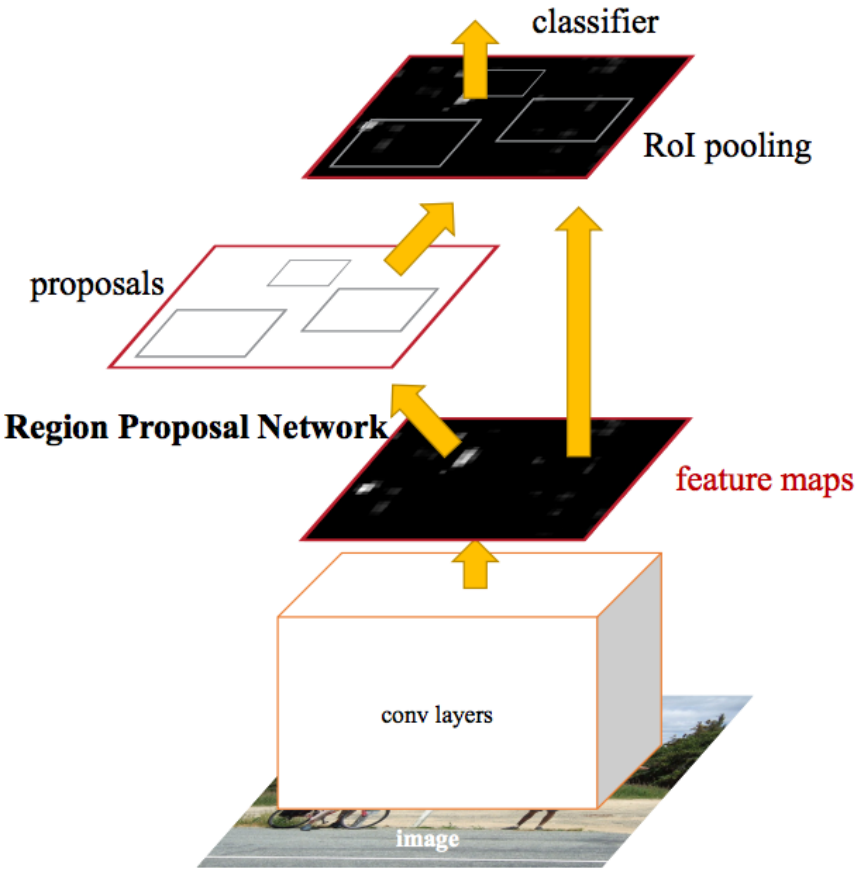
R-CNN は精度は高いのですが，上のそれぞれの段階を別々に学習させる必要があり，また，候補領域ごとに CNN による処理を行うため実行に非常に時間がかかるという欠点がありました．

Faster R-CNN

Faster R-CNN [S. Ren et al. 2015] は，R-CNN を改良したアルゴリズムであり，以下のような特徴があります．

- Selective Search の代わりに，入力画像に対して CNN を行って得られる特徴量から候補領域が学習可能な操作により提案されます（Regional Proposal Network）．候補とされた領域は　RoI pooling というプーリング操作により同じサイズの領域へ変換され，それぞれクラスと矩形位置の推定が行われます．このことにより，候補領域ごとに CNN を行う必要がなくなりました．
- 候補領域の選び方も含めて，すべての段階におけるコストを足したものをコストとするマルチタスク学習とすることで end-to-end（一度の学習でモデル全体を最適化する）な学習が可能となりました．

なお，直前に提案された Fast R-CNN もほとんど同様の構造を持つモデルですが，特徴量から候補領域を提案に既存の Selective Search を用いていて，この部分は別に学習させる必要があります．



Faster R-CNN

https://arxiv.org/abs/1506.01497

6.2 YOLO と SSD

YOLO

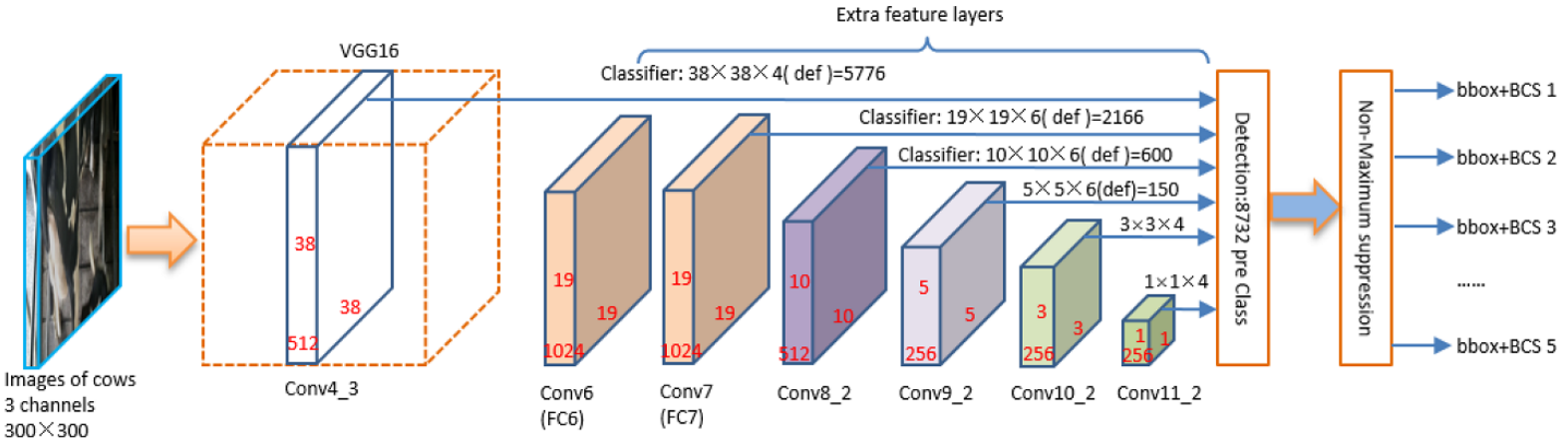
YOLO (You Only Look Once) [Redmon, J. et al., 2015] では，

1. 予め画像全体をグリッド分割しておく
2. 各セルごとに 1 つのクラスを推定する
3. それを用いて矩形領域を求める

とすることで，高速なアルゴリズムを実現しています．ただし，一つのセルに属する矩形領域は 2 つまでとされているので多くの物体が写っている画像の処理には向きません．

SSD

SSD (Single Shot MultiBox Detector) [Liu, W. et al., 2015] は，YOLO と似たようなアルゴリズムを用いることで Faster R-CNN よりも高速でありながら，CNN の各層における特徴マップに対して矩形領域を提案することにより，様々なスケールの物体を認識できるアルゴリズムです．また，end-to-end な学習も可能であり，Faster R-CNN とならんでよく利用されています．



SSD の構造

X. Huang et al. Animals 2019, 9(7), 470; https://doi.org/10.3390/ani9070470

参考ページ：

https://www.slideshare.net/takanoriogata1121/ssd-single-shot-multibox-detector-eccv2016

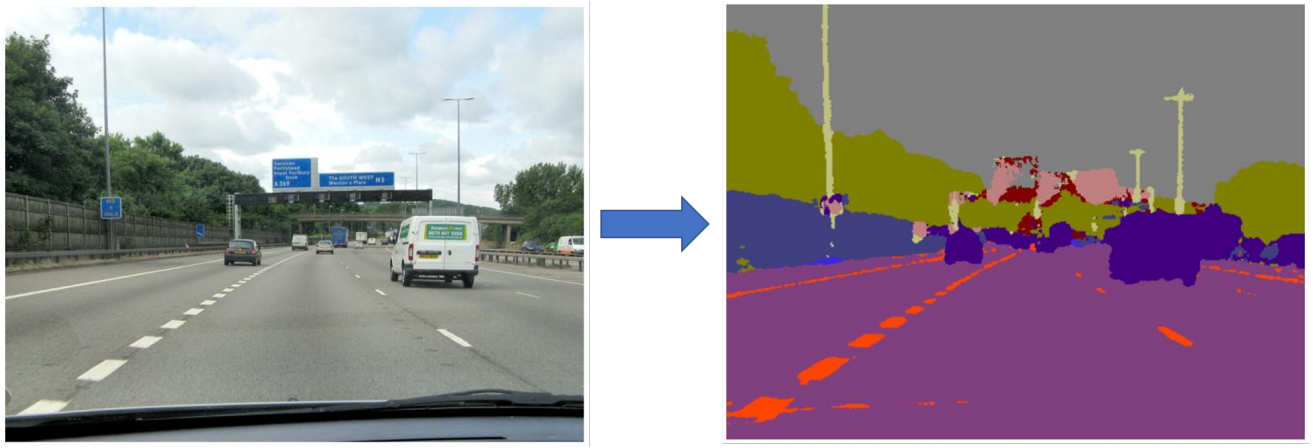
6.3　セマンティック・セグメンテーション

与えられた画像に対して，画像内の個々のピクセルがどのクラスに属するか分類するタスクをセマンティック・セグメンテーションといいます．これは自動運転や医療画像の分野において重要になります．

代表的な深層学習モデルとして以下の 2 つが挙げられます．

FCN　Fully Convolutional Networks:U-Net と似た構造を持つ

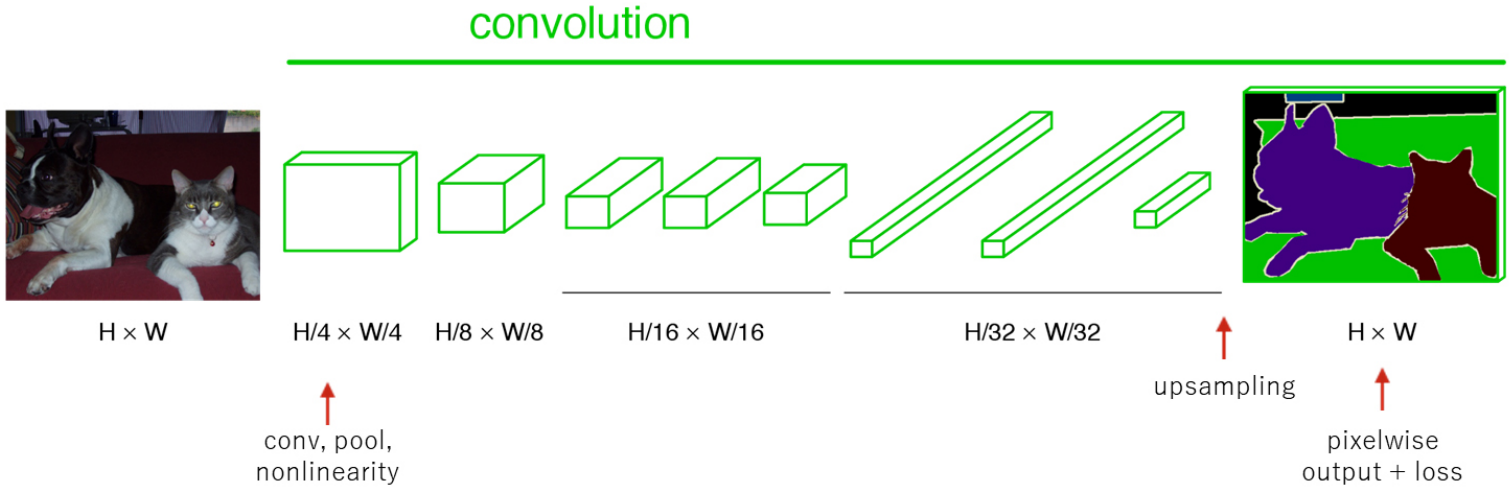
SegNet　エンコーダからデコーダにプーリングインデックスを伝える



https://mi.eng.cam.ac.uk/projects/segnet/

FCN: Fully Convolutional Networks

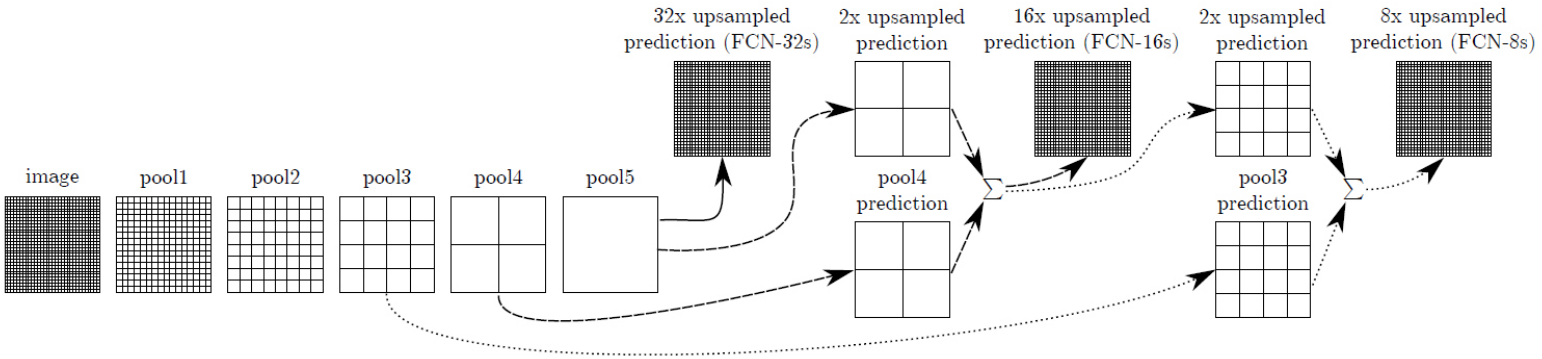
FCN 以前のモデルでは，ネットワーク内に全結合層があるために固定サイズの画像しか扱うことができませんでしたが，FCN では全結合層の代わりにフィルターサイズ 1×1 の畳み込み層を用いることで，画像サイズを固定する制約がなくなりました．



FCN の構造　https://arxiv.org/abs/1411.4038

FCN のデコーダ

FCN では CNN を施して得られる特徴マップに対して，その「ピクセル」毎にクラス分類を行い，その結果を逆畳みこみを用いてアップサンプリングすることで，元の画像についてピクセル単位でのクラス分類を行います．また，スキップ接続を通して CNN の各層における結果を特徴マップとして用いることにより，細かい境界の特定を行えるようになりました．



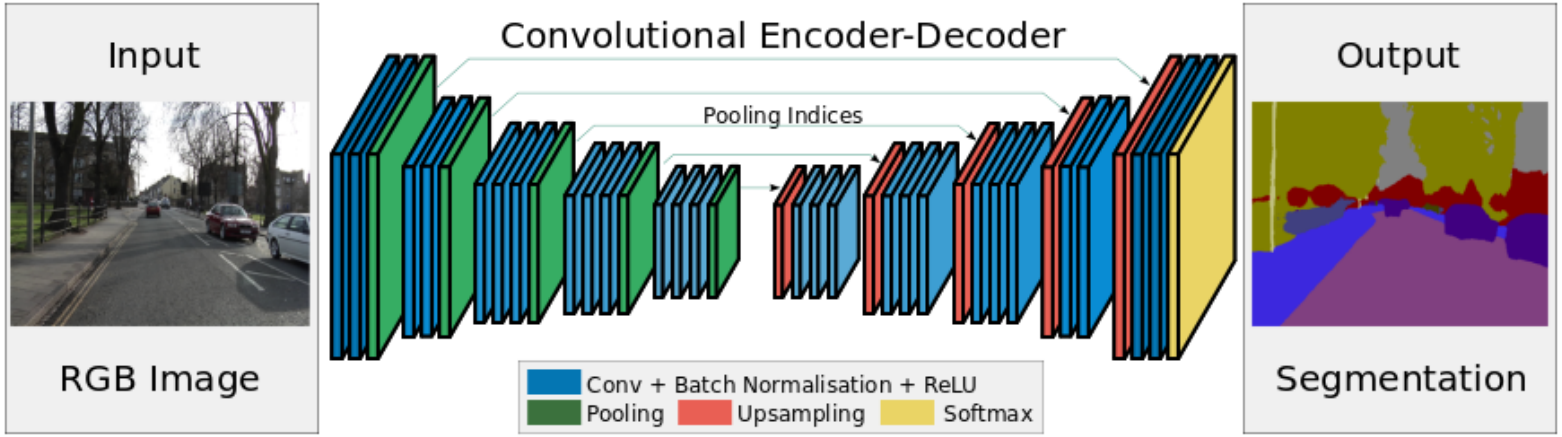
FCN のアップサンプリング　https://arxiv.org/abs/1411.4038

6.4 SegNet

セマンティックセグメンテーションのためのアーキテクチャは FCN 以後に様々なものが提案されてきましたが，SegNet [V. Badrinarayanan et al., arxiv.org:1505.07293] というモデルでは，エンコーダからデコーダに渡される情報を小さくすることでメモリ使用の効率化を図っています．エンコーダからデコーダに渡される情報を FCN と比較すると以下のようになります．

FCN 各段階の特徴マップ

SegNet 最終的な特徴マップと各マックスプーリング層で最大値をとったインデックス

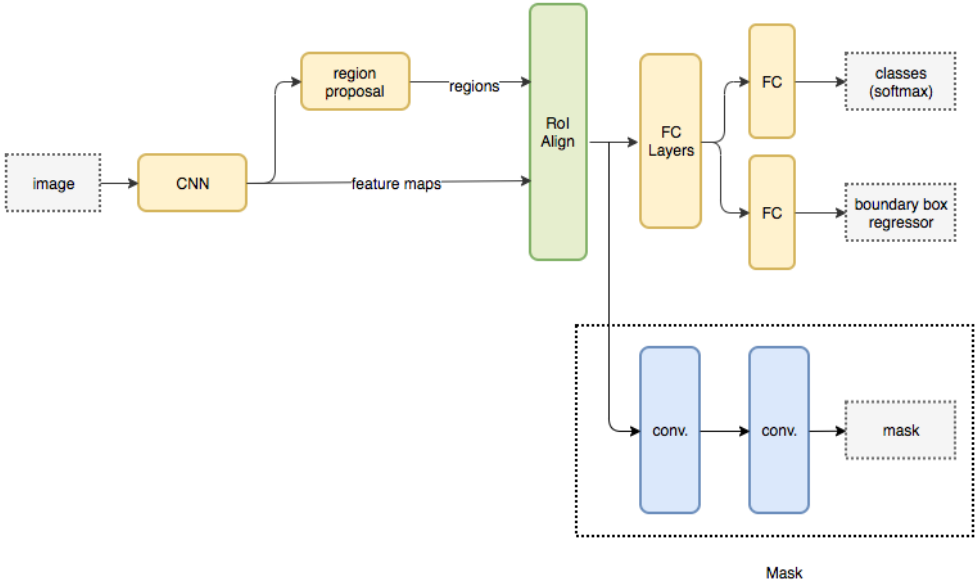


SegNet のアーキテクチャ　https://arxiv.org/abs/1505.07293

6.5 Mask R-CNN に関する演習

使用するノートブック: `3-10_MaskRCNN.ipynb`

Mask RCNN とは，基本的には Faster RCNN にセマンティック・セグメンテーションタスクを付け加えた一般物体認識用のモデルですが，中間層におけるバウンディングボックスの提案方法が改良されています．[K. He, et al. 2017]．このノートブックでは Keras を用いて Mask RCNN を実装したライブラリーを利用して動画処理を試みます．今回利用する実装は matterport 社が github で公開しているものです．



Mask R-CNN のアーキテクチャ

`https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272`