Deep Learning: Part 3

東京海洋大学 TUMSAT

竹縄知之 Tomoyuki Takenawa

目次

1 Convolutional neural networks

In this section, we study Convolutional Neural Networks (CNNs), which are the most basic method of deep learning.

CNN is a neural network that is effective for 1D, 2D, and sometimes 3D grid data, and mainly consists of ”convolutional operations” and ”pooling operations”. In this section, we use 2D image data as an example.

In the ImageNet Large Scale Visual Recognition Competition (ILSVRC) held in 2012, a CNN-based method called AlexNet won the competition by a wide margin over previous approaches that used support vector machines, thus bringing deep learning to the forefront of attention.

1.1 Convolution operation

For a single two-dimensional data without channels

A convolution operation (or a convolution process) is an operator that, for example, for

Input $X =$

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

Filter $F =$

1	1	1
2	0	1
−1	−1	0

(the gray area is the area used to calculate y_{00} below), calculates Y as

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

and thus$Y =$

4	8
20	24

.

T hat is, the operation

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

is done as

- y_{00} is the "inner product" of the filter and the part of the same shape in the upper left corner of X.
- y_{01} is the part taken from X shifted by 1 to the right.
- y_{10} is the one shifted downward.
- y_{11} is shifted down by 1 to the right by 1.

We denote convolution as

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

by using $\circledast$ symbol.

Formal definition and shape of the output

When the shape of the input data X is $\mathrm{H} \times \mathrm{W}$ and the shape of the filter F is $\mathrm{FH} \times \mathrm{FW}$, the output data Y is a matrix of shape $(\mathrm{OH}, \mathrm{OW}) = \big((\mathrm{H} - \mathrm{FH} + 1), (\mathrm{W} - \mathrm{FW} + 1)\big)$ whose (o_h, o_w) component is

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

. Here, for an array A, $A[i,j] = a_{i,j}$ denotes its (i,j) component.

Characteristics of convolution operations

Sparse connectivity: Neurons in neighboring layers are more likely to be unconnected.

Parameter sharing: Use the same filter regardless of position

From these, it is possible to extract information from the image step by step.

1.2 Extensions for actual application

When we actually use convolution in a neural network, we use a slight extension of the basic operations.

Bias

Adding a constant to the output result all at once is called a bias. For example,

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

Padding

Filling a constant (such as 0) around an input X is called padding. In the case of convolution operations, the constant is padded with 0s of the same width. By padding, for example, it is possible to make the output the same shape with the input.

For example, if we padded the area around it with one row of zeros, we have

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

Strides

The number of cells to be shifted can be changed to an integer of two or more. This integer is called the stride.

For example, if we take the stride as two in

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

since the filter is applied with a shift of 2 cells, the size of the output becomes 3×3 as

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

Calculate it by yourself!

Formula

When the padding size is P and the stride is S, the output size is OH $\times$ OW, where

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

Here, in the case of non-divisible, the fractional part shall be truncated. In Python like coding, this is written as

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

Problem: Show the equation above.

1.3 Convolution operation (with channels and mini-batches)

The convolution operation we have seen so far is an operation that takes as input a second-order array consisting of vertical and horizontal directions. In practice, however, it is necessary to deal with a four-order array that includes the channel direction and the direction of the mini-batch data number. Here, channels are used to maintain the diversity of data, like RGB (Red, Green, Blue) in color images.

The input/output data array has four dimensions

Let

N : Size of the minibatch	
C: Number of channels of input	OC: Number of channels of output
H: Height of input data	FH: Height of the filter
W: Width of input data	FW: Width of the filter

then shapes of input X, filter F, and output Y to the convolutional layer are

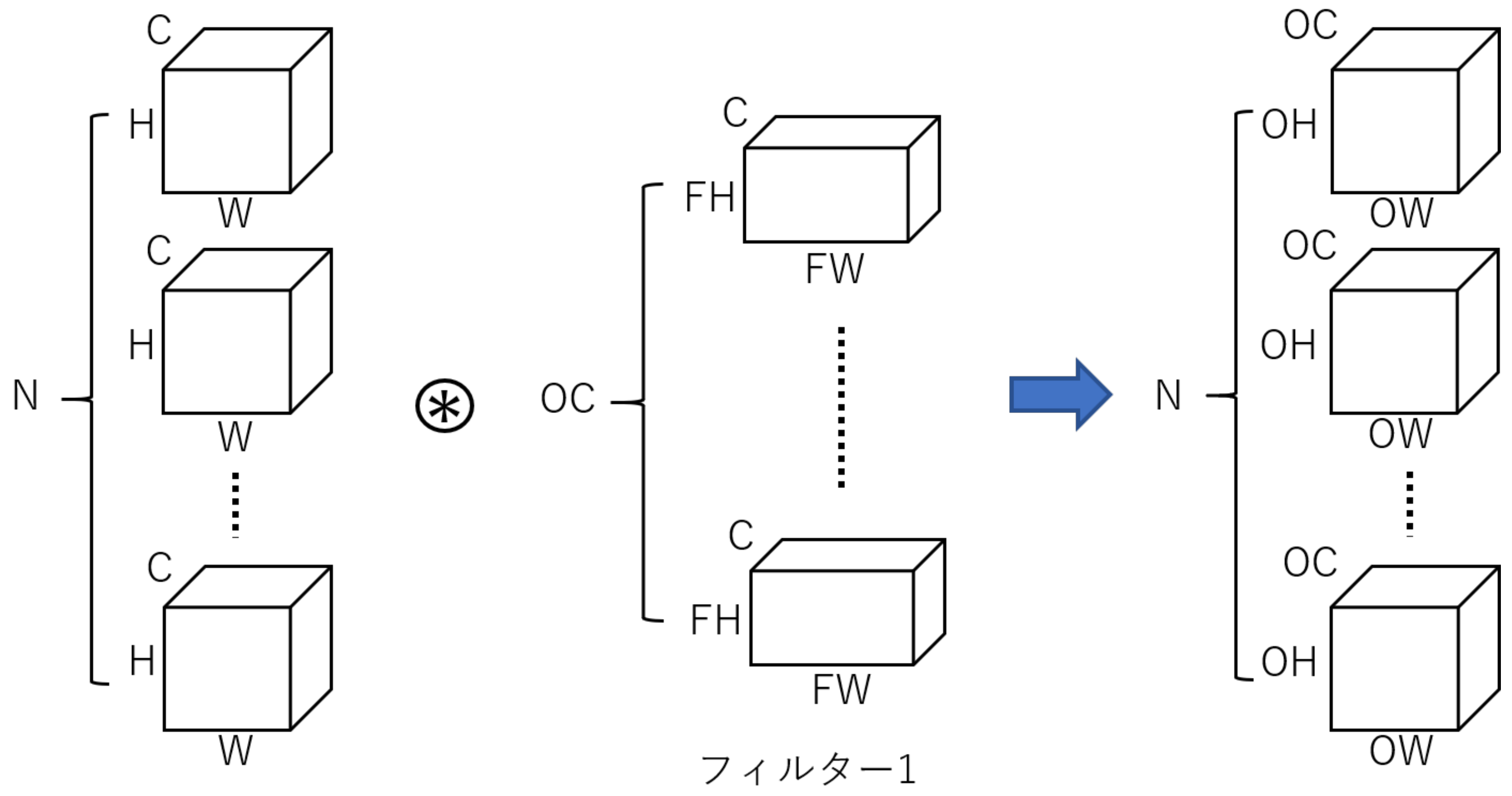
$$\begin{aligned} X &: (\mathrm{N}, \mathrm{C}, \mathrm{H}, \mathrm{W}) \\ F &: (\mathrm{OC}, \mathrm{C}, \mathrm{FH}, \mathrm{FW}) \\ Y &: (\mathrm{N}, \mathrm{OC}, \mathrm{OH}, \mathrm{OW}) \end{aligned}$$

respectively, where the height and the width of output are given by

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

(P: padding, S: stride).

Illustration of the dimensions of convolutional operations



im2col

When implementing the convolutional operation, the bottleneck is to extract the required number of vectors of the same shape as the filter from the input X.

For example, from Input $X = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array}$, we need to extract arrays of shape $2 \times 2 \times 3 \times 3$:

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

A simple implementation would be to write the following using the for statement for the OW $\times$ OH $= 2 \times 2$ dimensions of the output.

```
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 (continued)

However, this method requires the for statement to be calculated OH $\times$ OW times, and when X becomes large, the process will take a long time.

Hence, by shifting 2×2 (shape of the output data) part of X by 3×3 (shape of the filter) times, we first construct an array of shape $3 \times 3 \times 2 \times 2$:

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

and then, by arranging the numbers enclosed in squares, we extract the $(0,0)$ component of the array of shape $2 \times 2 \times 3 \times 3$.

im2col (continued)

The second operation is to rearrange $\hat{X}$ as $\tilde{X}[o_h, o_w, f_h, f_w] = \hat{X}[f_h, f_w, o_h, o_w]$, so the algorithm can be written as follows.

```
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) #transposition
```

This method is efficient because the loop of the for statement only runs FH $\times$ FW times: the filter size. An implementation using this algorithm is called im2col.

1.4 Exercises on the convolution operation

Notebook: `3-1_畳み込み演算.ipynb`

In this notebook, we will do an exercise on the convolution operation. Basically, we will implement in NumPy and verify what we have shown in the slides so far. A simple problem is given at the end.

Please note the followings.

- Implementation of im2col, especially the way to calculate $\tilde{X}$ and $\hat{X}$.
- Handling of fourth-order arrays
- Slices of arrays

1.5 Pooling

Pooling is an operation that obtains second-order data with a smaller dimension from second-order input data. Unlike convolutional operations, the filter has no parameters.

The following is an example of a MaxPooling operation of frame size 2×2 on an input X of shape 4×4.

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

The $(0, 0)$ component '6' of the output Y is the maximum value of the $\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}$ part of X. Similarly, the $(0, 1)$ component '8' is the maximum value of the $\begin{array}{|c|c|} \hline 3 & 4 \\ \hline 7 & 8 \\ \hline \end{array}$ part of X. The same is true for the $(1, 0)$ and $(1, 1)$ components.

Pooling (continued)

In this way, the inner product of the filter F in the convolution operation is replaced by the operation of ”taking the maximum value” in pooling. Also, to avoid overlapping of the parts of X that take the maximum value, the size of the frame and the width of the stride are taken to be equal. Thus, the frame is always a square (if the stride is also changed horizontally and vertically, a rectangular filter is possible, but it is not often used).

In the actual calculation, it is passed through $\tilde{X}$ as in the convolution operation.

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

Support for channels and mini-batches

Pooling is done on a per-channel basis. Therefore, the number of channels does not change before and after the layer. Of course, the size of the mini-batch does not change either. Other than that, it is almost the same as the convolutional layer.

Note: Pooling has no parameters. Max pooling with frame size $S \times S$ and convolution with filter whose height, width and stride are S have the same output size, but differ in whether they use learnable filters or not, and whether they sum over input channels or not. (In the case of convolution, the filter is learnable and sums over the input channels.)

1.6 Exercises on pooling operations

Notebook: `3-2_プーリング演算.ipynb`

In this notebook, we will do exercises on (Max) pooling operations. As in the case of convolution operations, we will implement and verify what we have shown in the slides so far in NumPy. At the end, there is a simple problem.

Please note the following.

- Implementation of im2col, especially how to calculate $\tilde{X}$ and $\hat{X}$.
- Handling of fourth-order arrays
- Slicing of arrays

2 Implementation of convolutional neural networks

In order to implement CNN, Backpropagation in the convolutional layer should be implemented as a Class.

However, Backpropagation of the convolutional layer is quite complicated.

2.1 Convolution layer

Backpropagation♯

Let's consider an example with

$$\text{Input } X \circledast \text{Filter } F = \begin{array}{|c|c|c|c|}\hline x_{00} & x_{01} & x_{02} & x_{03} \\ \hline x_{10} & {\color{red}x_{11}} & x_{12} & x_{13} \\ \hline x_{20} & x_{21} & x_{22} & x_{23} \\ \hline x_{30} & x_{31} & x_{32} & x_{33} \\ \hline \end{array} \circledast \begin{array}{|c|c|}\hline f_{00} & f_{01} \\ \hline f_{10} & f_{11} \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline y_{00} & y_{01} & y_{02} \\ \hline y_{10} & y_{11} & y_{12} \\ \hline y_{20} & y_{21} & y_{22} \\ \hline \end{array} = Y$$

ans Error $E = E(Y)$. Then, since ${\color{red}x_{11}}$ appears in 4 components of

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

we can calculate as

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

Backpropagation (continued)

To simplicity, we write as $\dfrac{\partial E}{\partial y_{ij}} = dy_{ij}$, $\dfrac{\partial E}{\partial x_{ij}} = dx_{ij}$. Similar to the above, we can calculate as

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

In the implementation, this calculation is done efficiently via

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

Backpropagation (continued)

Derivative with respect to parameter F: And f_{00} appears in 9 components of

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

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

we can calculate as

Backpropagation (continued)

Similarly, we obtain

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

This also can be implemented more efficiently by using

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

2.2 Exercises on convolution layers

Notebook: `3-3_畳み込み層.ipynb`

In this notebook, we implement a convolution layer and compare it with numerical differentiation.

Please note the following

1. Operations on higher-order arrays
2. How to use im2col: The im2col algorithm is used in the code
3. Shapes as an array of inputs and outputs

2.3 Pooling layer

Backpropagation

If the propagation of Pooling layer is

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

$$\to Y = \begin{array}{|c|c|}\hline y_{00} & y_{01} \\ \hline y_{10} & y_{11} \\ \hline \end{array} = \begin{array}{|c|c|}\hline \textcolor{red}{x_{11}} & \textcolor{red}{x_{13}} \\ \hline \textcolor{red}{x_{20}} & \textcolor{red}{x_{22}} \\ \hline \end{array} \to E$$

(the red letters are the maximum in each single block in $\tilde{X}$), the backpropagation is given by

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

2.4 Exercises on pooling layers

Notebook: `3-4_プーリング層.ipynb`

In this notebook, we implement a pooling layer and compare it with numerical differentiation.

Please note the following. Also, solve the problem at the end of the notebook.

1. Difference from convolution layers
2. The function that takes the maximum value of an array: `numpy.max()` is used
3. Differentiation of a function that takes the maximum value of an array: `numpy.argmax()` is used

2.5 Exercises on implementation of convolutional neural networks

Notebook: `3-5-1_CNN.ipynb`

In this note, we will implement convolutional neural networks using the layers we have implemented so far.

Please note the following. Also, solve the problem at the end of the notebook.

1. Shapes of data arrays flowing through
2. Computational complexity (computation time)
3. Accuracy

2.6 Exercise on the use of GPU

Notebook: `3-5-2_CNN_GPU.ipynb`

GPU

As you can see from the exercises in the previous section, the computational complexity of CNNs is very large and the computation is time consuming. In order to solve this problem, GPU (Graphics Processing Unit) is used for array computation instead of CPU (Central Processing Unit) which is used for general computation. GPUs are originally designed for fast computation of 3D graphics rendering and image processing, and they can compute arrays much faster than CPUs.

In particular, deep learning involves a lot of matrix computation, which can be accelerated by using GPUs. In order to use GPUs for array computation, it is common to use the CUDA (Compute Unified Device Architecture) platform (programming language) developed and provided by NVIDIA.

Of course, TsnsorFlow and PyTorch also support CUDA, but there is a library called CuPy provided by Preferred Networks (a Japanese company) that allows you to use CUDA in a similar way to NumPy. In this exercise, we will use CuPy.

3 Use of CNNs

In this section, we will summarize how CNNs are used and introduce some innovations in the use of CNNs.

By appropriately designing the output according to tasks, CNNs can be used not only for image regression and classification, but also for general object recognition, generative modeling, recurrent convolutional neural networks, and many other situations. Such a way of designing the output of the network is called structured output.

3.1 Types of data

Depending on the type of data, the dimension (number of orders) of one data item and the existence of channels change. The followings are typical examples. (The direction of the batch is not counted.)

Single-channel	multichannel	
1D	Audio waveforms, DNA sequences	Skeleton animations (skeleton movement)
2D	Monochrome images Fourier transformed audio data	Color images
3D	Monochrome videos 3D density data (CT scan)	Color videos

As was the case with MNIST handwritten digit data recognition, it is common to use multiple channels in the intermediate layer even if the input is a single channel.

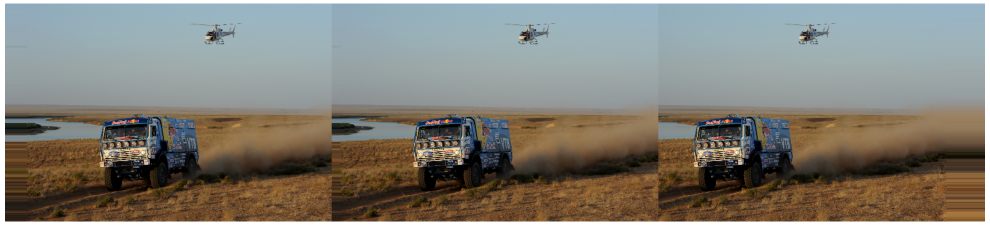
3.2 Data augmentation

A method to increase the training data by processing the original images, for example

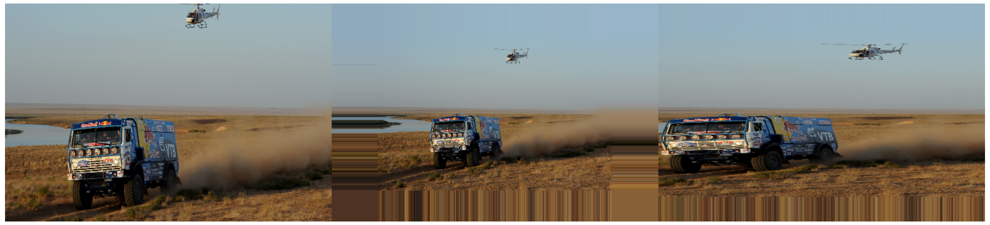
https://upload.wikimedia.org/wikipedia/commons/6/6f/Silk_Way_Rally_2011.jpg

All of them are applied randomly.

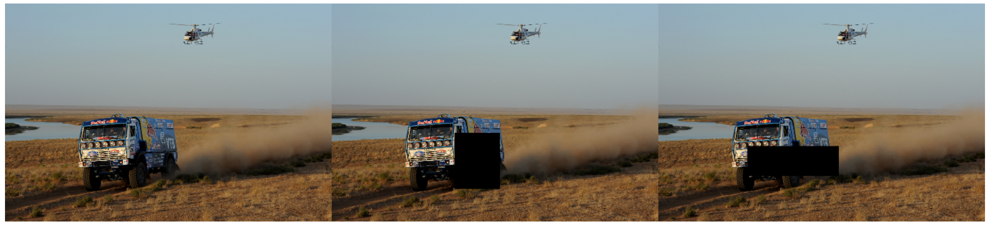
Shift:



Zoom in and Zoom out:

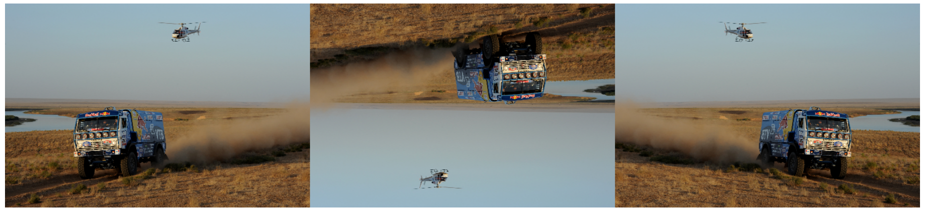


Cut:



Data augmentation (continued)

Flip:



Rotate:

Brightness:

Shearing (some linear transformaton) :

However, in the case of the photo above, for example, it is unlikely that the image is upside-down, so do not flip the image upside-down in this case. Data augmentation is intended to improve the generalization performance of the model by making the training data's distribution closer to the original data's distribution.

3.3 Exercises on data augmentation

Notebook: `3-6_Augmentation.ipynb`

In this notebook, we train a convolutional neural network using data augmentation on a color image dataset called Cifar10. We compare three implementations: one using NumPy-CuPy, one using Keras-TensorFlow, and one using PyTorch.

1. Difference between accuracy on training data and accuracy on test data
2. Where is the bottleneck of computation: Convolutional operations are also computationally demanding, but in fact data augmentation is also time-consuming because it requires processing each image one by one. In order to avoid this, there are several ways to do this, such as increasing the data separately from the training in advance, or increasing the data on a different CPU in parallel with the training.

3.4 Transfer of features

If we have a learned model that has been trained for one task, the method of using the learned model for another related task is called transfer learning. For example, if a model has been trained to classify images of dogs, cats, horses, and sheep, and now we wish to add cows to the list, instead of starting from the beginning, we can use some of the learned parameters (in this case, the layers up to the middle) as initial values.

- Using a model trained on a task with a large amount of data for a task with only a small amount of data available for training
- Multitask learning (learning several related tasks at the same time, such as image object recognition and classification) is another example.
- Using a self-encoder or similar to perform unsupervised learning beforehand and then re-training a part of the model with supervised data is another example.

One-shot learning and zero-shot learning

Extreme examples of transfer learning are one shot learning and zero shot learning.

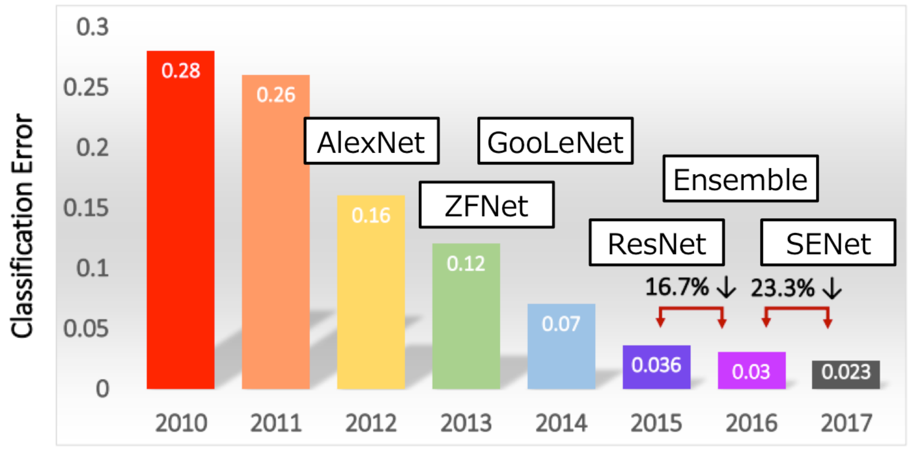
One-shot learning Literally, a method of retraining only one data at a time. As shown in the example above, this method is used when learning to classify images of dogs, cats, horses, and sheep, and then adding cows. In order to perform one-shot learning in deep learning, the model must be such that it originally outputs a feature representation, as described in [G. Koch etal. “Siamese neural networks for one-shot image recognition” ICML Deep Learning Workshop. Vol. 2. 2015.] .

Zero-shot learning By adding data other than images, such as text, to the learning process, we can classify cows without learning images of cows, based on information such as ”as big as a horse, slightly round, no mane”.

4 Evolution of CNN

In the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC), a CNN-based method called AlexNet won the competition by a large margin, and deep learning has been attracting attention ever since. Since then, CNNs have become the mainstream in image recognition, and new models have been proposed every year.

Among them, Residual Networks (ResNet), the winning model of ILSVRC in 2015, introduced ”skip connections” between distant convolution layers to achieve learning with very deep networks.



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

Evolution of CNN (continued)

In this section, we will introduce the followings.

- VGG and GoogleNet: 2nd and 1st place in 2014 ILSVRC, respectively
- Residual Networks: skip connections are introduced to achieve deep networks
- Dense Networks: CNN with dense skip connections (note that this is not a ”Dense Neural Networks: all connected neural network”)
- Mobile Networks: Networks with efficient convolutional operations

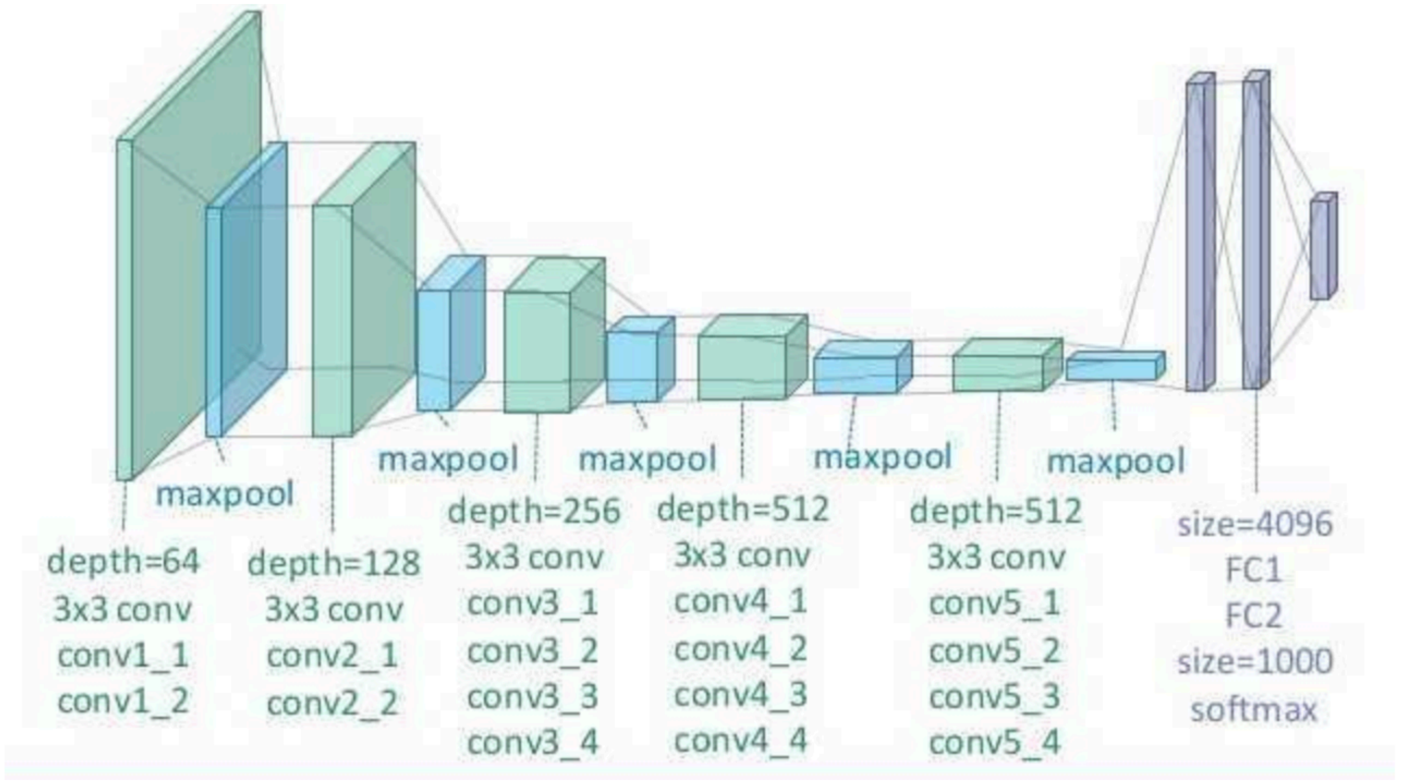
CNNs are widely used not only for image recognition, but also as a base network for solving various tasks such as segmentation and object detection. CNNs are also used in the fields of natural language processing, speech processing, game AI, etc., and occupy an important position in neural networks.

Reference: 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 and GooglNet

VGG

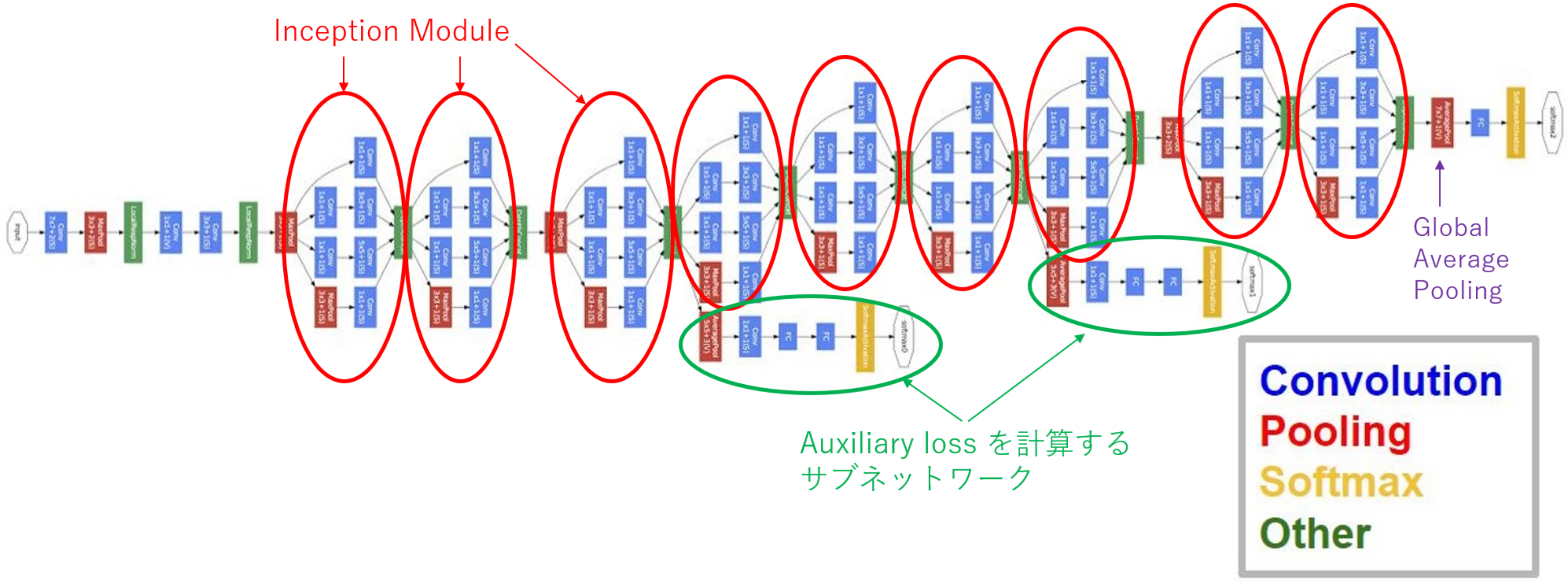
VGG [Simonyan, K., Zisserman, A., 2014] is a model that has been used as a template for the CNNs we have built in this course. As shown in the figure below, the convolutional layer and pooling layer are stacked, and the final output is the fully connected layer.



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

GoogLeNet

GoogLeNet [Szegedy, C. et al., 2014] has a structure of CNNs connected by branches as shown below; the difference with ResNet is that there are no connections that do nothing.



[Szegedy, C. et al., 2014]

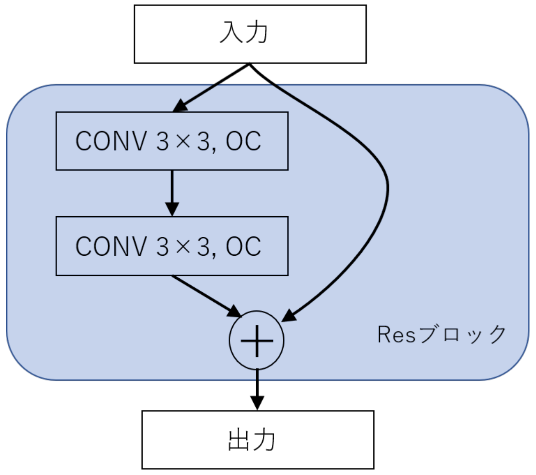
4.2 Residual Networks

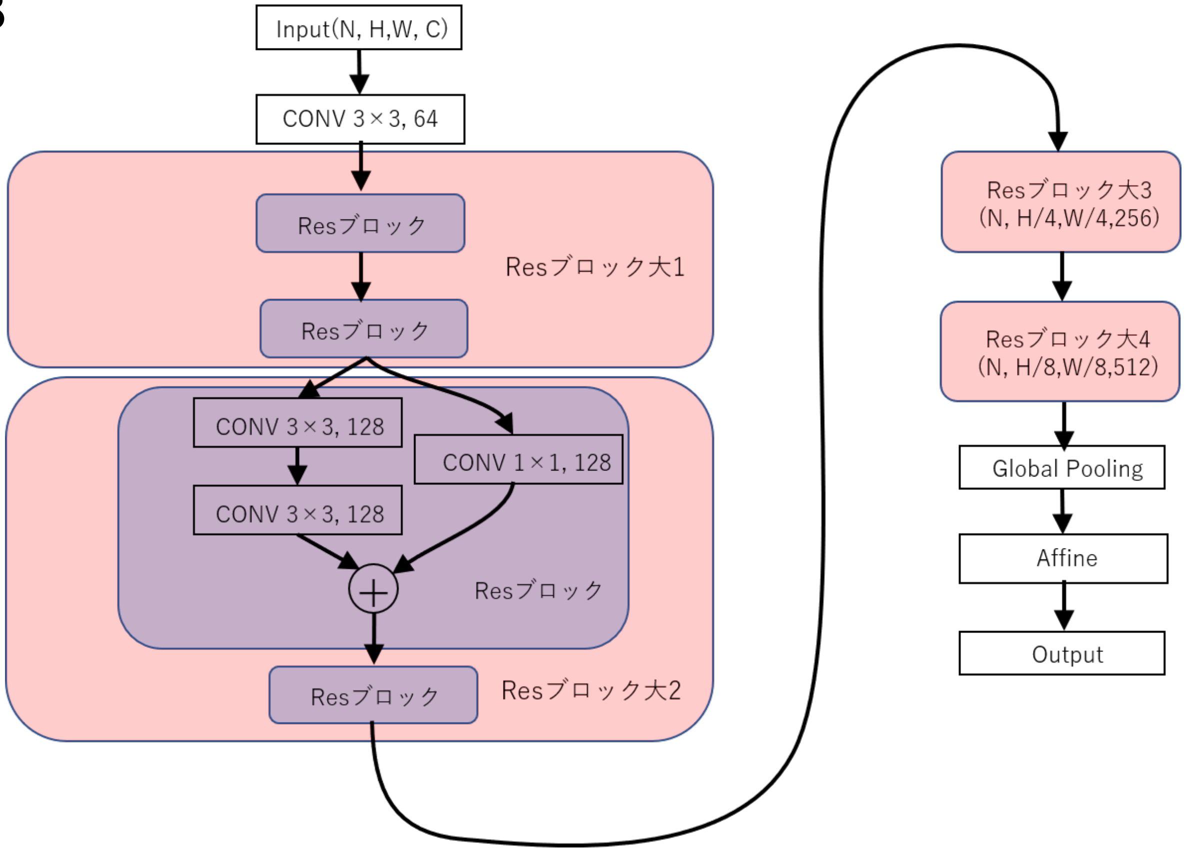
In neural networks that propagate through the layers in order, such as all-connected networks and convolutional networks, as the layers become deeper, information such as the derivative decreases or increases in exponential order with respect to the number of layers, and thus becomes difficult to propagate.

For example, if the derivative is doubled for each layer, then $2^{10} = 1024$ times for 10 layers, $2^{20} = 1,048,576$ times for 20 layers, and so on.
Although appropriate initial values for the parameters and batch normalization have been used to deal with this problem, when the layers become too deep (e.g., more than 30 layers counting only all the coupled layers or convolutional layers), the information is buried in random numbers and computational errors, and it is still difficult to communicate.

Residual Networks (ResNet) are neural networks proposed by Kaiming He et al. in 2015 to deal with this problem by adding an ”edge” to the convolutional network that skips some layers and conveys information [https ://arxiv.org/abs/1512.03385].

Structure of ResNet18





Stack several of the smallest blocks in the left diagram to make a whole.

$\boxed{\text{CONV } 3 \times 3, \text{ OC}}$ is a convolution layer with filters of shape 3×3 and OC=C output channels.

In blocks 2, 3, and 4, the first Res block halves the image size by setting the stride to 2.

For the other convolution layers, the stride is 1 and padding is used to keep the image size unchanged.

The first convolution depends on the input data.

The output size for Global Average Pooling is (N, C).

Add

The $\oplus$ on the previous page is simply an addition layer that outputs

$$Z = X + Y$$

for inputs X and Y. The X and Y must be arrays of the same shape.

Backpropagation

Backpropagation of $\oplus : Z = X + Y$:

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

Backpropagation at the brunch $X \to X_1, X_2$:

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

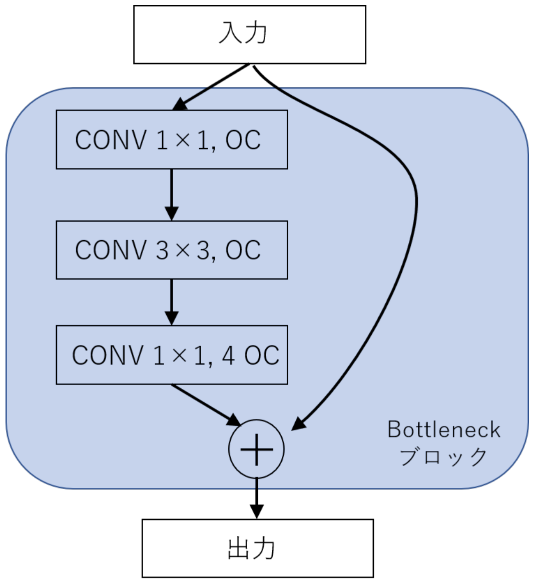
Typical ResNets

ResNets are classified by “number of output channels × number of small blocks” in large blocks.

type	ResNet18	Resnet 34	Resnet50	Resnet101	ResNet152
Large block 1	64×2	64×3	b64 $\times 3$	b64 $\times 3$	b64 $\times 3$
Large block 2	128×2	128×4	b128 $\times 4$	b128 $\times 4$	b128 $\times 8$
Large block 3	256×2	256×6	b256 $\times 6$	b256 $\times 23$	b256 $\times 36$
Large block 4	512×2	512×3	b512 $\times 3$	b512 $\times 3$	b512 $\times 3$

Here, b62, b128, etc. are blocks called “bottleneck”.

Bottleneck bx, (x = OC):



4.3 Exercises on ResNets

Notebook: `3-7_ResNet.ipynb`

In this notebook, we train ResNet using data augmentation on a color image dataset called Cifar10. The implementation is done using Keras-TensorFlow.

Note that the notebook also implements Wide ResNet, which was proposed to improve performance by increasing the number of channels in a less deep network, while ResNet was designed to improve performance by increasing the number of layers. Wide ResNet is designed to improve performance by increasing the number of channels even in networks that are not that deep. [Zagoruyko-Komodakis, "Wide Residual Networks", https://arxiv.org/abs/1605.07146]

1. Generating a network using Model() in Keras
2. Save and load trained parameters
3. Visualizing the model
4. How long does it take to train?

5 Generative models

In this section, we will learn about neural networks called generative models. A generative model is a model that generates new data by imitating learned data such as images and sounds.

5.1 Generative and discriminative models

Generative models A model that learns training data and generates new data similar to those data is called a generative model. The model is trained so that the distribution of the generated data matches the distribution of the training data (the distribution of the data is the probability of occurrence of each individual data). Most generative models consist of encoders and decoders, where the encoders obtain a latent representation and the decoders generate new (or the same as the input) data.

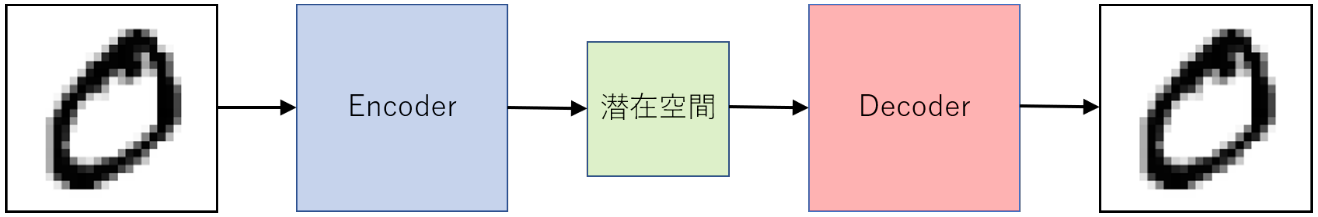
Discriminative models A model that determines whether the given data is training data or data generated by a model is called a discriminative model.

5.2 Autoencoders

Autoencoder is a neural network that combines two neural networks, Encoder and Decoder, and it is capable of grasping the characteristics of the target data by performing dimensionality reduction by the encoder, and generating new data by adding noise to the middle layer (latent space).

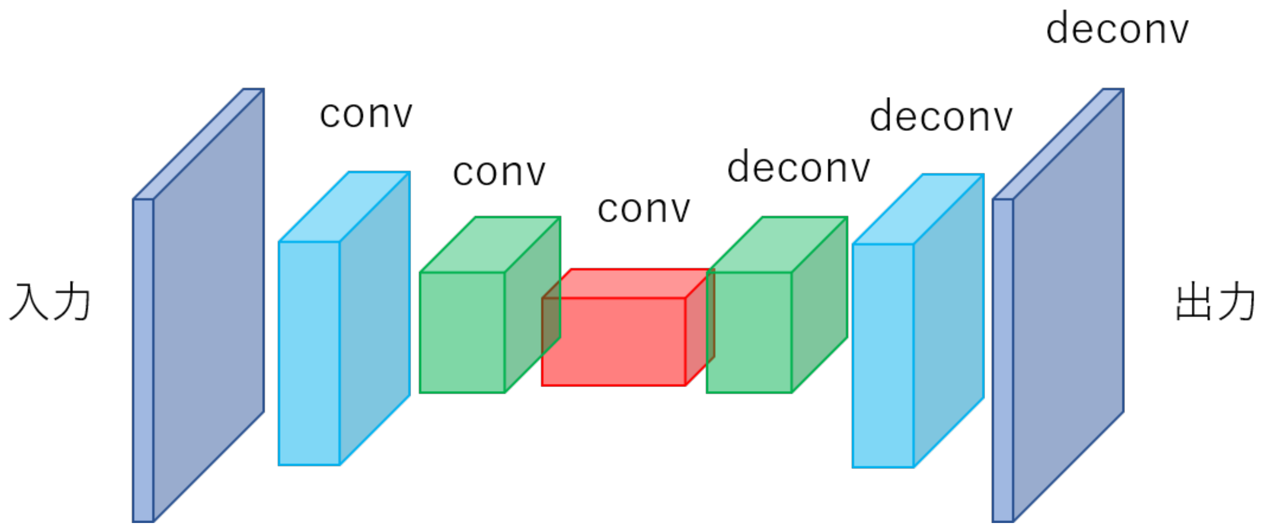
The loss function of autoencoders usually uses the sum of the squared errors between the input and output data. This helps to train the input and output data to follow the same distribution. When the input data is a real vector with no range constraint, the identity map is often chosen as the activation function for the output layer, and when there is a range of values, such as for image data, the sigmoid function is used.

Autoencoders can also be used to detect anomalous values. In this case, anomaly is detected when the difference between the input data and output data is large.



5.3 Deconvolution (transposed convolution) and convolutional autoencoders

Autoencoders originally consisted of a stack of affine layers (fully connected layers), but nowadays, convolution operations are commonly used. However, the problem is that the convolution operation makes the image size the same or smaller, not larger. Considering the purpose of autoencoders that is to obtain a latent representation, it is necessary to reduce the image size once (where increase in the channel direction is acceptable). Thus, an operation called ”deconvolution” or ”transposed convolution” (the two are the same) is used.



Convolutional autoencoder

Decomvolution operation

For Input $X = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$ and Filter $F = \begin{array}{|c|c|}\hline 1 & -1 \\ \hline 2 & 1 \\ \hline \end{array}$, the decinvolution with Stride 2 fistly dilate X as

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

and next calculate its conbolution with F as

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

Here, stride is a parameter that determines how much the elements of X are separated during the dilation.

Decomvolution operation (continued)

When stride is one, decomvolution is as follows

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

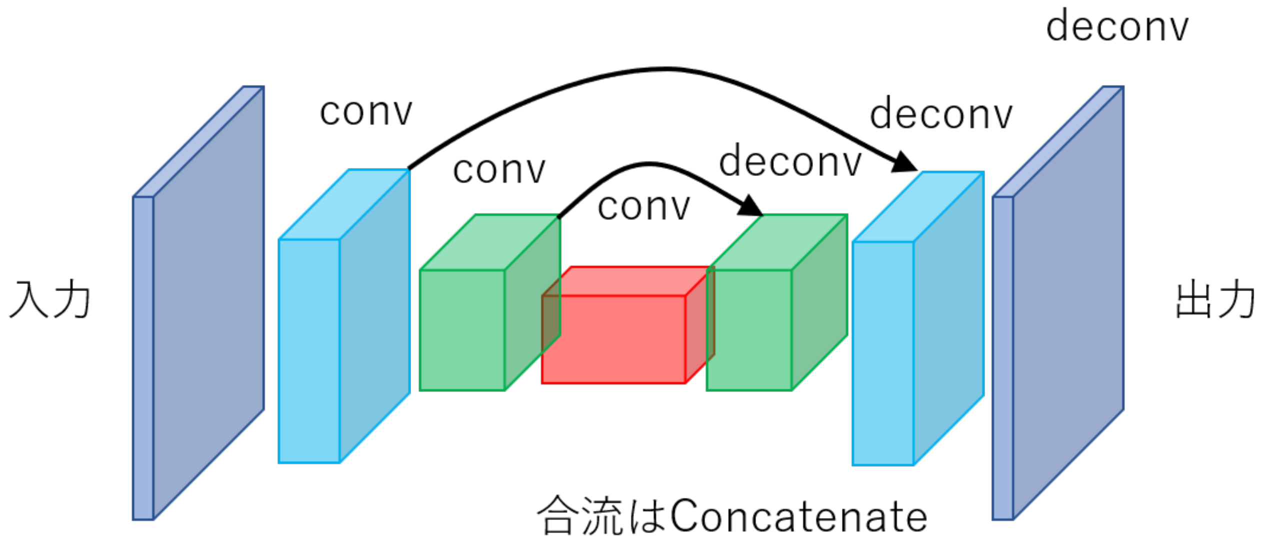
The amount of padding around the cells is determined by the size of the target.

Note that deconvolution is not an inverse operation of convolution, since convolution of Y does not go back to X. Rather, when convolution is written as matrix multiplication, it corresponds to multiplying a transposed matrix, so transposed convolution is more accurate.

Also note that the number of paddings in the deconvolution is the number of paddings needed to go from Y to X (or another matrix of the same size) in the convolution. For example, the two examples above both have 0 padding. There is a nice animation on the following site.
`https://github.com/vdumoulin/conv_arithmetic`

5.4 U-Net

Convolutional autoencoders can be applied to the task of transforming the style of an image. For example, a task called semantic segmentation that classifies the class of each pixel in an image. A model called U-Net, which adds skip connections to the convolutional autoencoder, is useful for such a task. FCN (Fully Convolutional Networks) has a similar architecture, but U-Net is simpler.



5.5 Exercises on segmentation using autoencoders (U-Net)

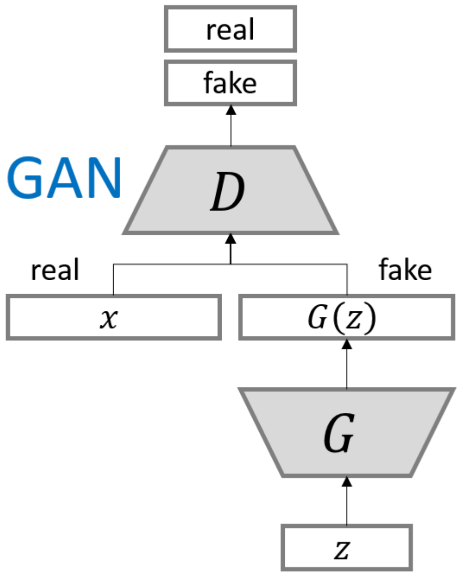
Notebook: `3-8_U-Netによるセグメンテーション.ipynb`

In this notebook, we implement U-Net, a convolutional autoencoder with skip connections, using Keras, and apply it to a task called semantic segmentation. Semantic segmentation is the task of predicting the class to which each pixel in an image belongs.

- The class of the transpose convolution layer in Keras is `Conv2DTranspose`.
- How does the size of the height and width and the number of channels change for layers of U-Net?
- Error function in semantic segmentation: cross entropy is used.
- Dataset: Identifying the left ventricle in an MRI image of the heart.
- Other applications?

5.6 GAN

Generative Adversarial Networks (GAN) [Goodfellow et al., 2014] is a model that alternately trains its Generator and Discriminator. The Generator takes as input a noise $\mathbf{z}$ generated by a random number and generates new data $G(\mathbf{z})$. The discriminator takes either the randomly selected training data $\mathbf{x}$ or the data generated by the generator $\mathbf{x} = G(\mathbf{z})$ as input and outputs the predicted probability $D(\mathbf{x})$ that it is the training data. That is, if $D(\mathbf{x}) > 0.5$, it predicts that $\mathbf{x}$ was the training data, and if $D(\mathbf{x}) \leq 0.5$, it predicts that $\mathbf{x}$ was the generated data.



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

Learning of GANs

GAN is a model that alternately

1. trains only the discriminator with the parameters of the generator fixed;
2. trains only the generator with the parameters of the discriminator fixed.

Here, the generator is trained to generate data with a distribution that is as close as possible to that of the training data, and the discriminator is trained to be highly accurate in discriminating against such data.

Such learning of GAN is similar to the structure of a forger and a connoisseur competing with each other to improve their skills in painting.

When learning is complete and the distribution $P_{\mathbf{data}}(\mathbf{x})$ of the training data and the distribution $P_{\mathbf{gen}(\mathbf{x})}$ of the generated data match, $D(\mathbf{x})$ becomes infinitesimally close to 0.5 because identification is impossible.

Note: For example, in the case of images, the distribution of data here refers to the probability distribution of the occurrence of each image. For instance, there are many images of dogs, but when we say the image of a dog, we mean the probability that each one of them appears.

GAN's value function

If the correct label y for the discriminator is $y = 1$ when the input is training data and $y = 0$ when the input is generated data, then the conditional log likelihood (cross-entropy $\times(-1)$) of the discriminator is

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

.

Therefore, the expected value of this when inputting the training data is (since $y = 1$)

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

and the expected value when inputting the generated data is (since $y = 0$)

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

Here, since the probability that the data generated will be $\mathbf{x} = G(\mathbf{z})$ is equal to the probability that the noise will be $\mathbf{z}$, it holds that

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

(Actually, it is a continuous distribution, but for simplicity, I explained it as a discrete distribution.)

GAN's value function (continued)

In particular, if we assume that the rate of input of training data is equal to the rate of input of data by the generator (this is how it is taken in GAN), then the 2 times conditional log likelihood of the discriminator is

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

$V(D,G)$ is called the value function of GAN.

The discriminator is trained to maximize this $V(D,G)$. Therefore, the training of the discriminator can be expressed as $\max_D V(D,G)$. On the other hand, the generator is trained to deceive the discriminator as much as possible, that is, to minimize $V(D,G)$. Therefore, the training of the generator can be expressed as $\min_G V(D,G)$. Thus, the learning of the GAN can be expressed as

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

. Since the first term of ① is independent of the noise $\mathbf{z}$ and the generator G, the learning of the generator can also be expressed as

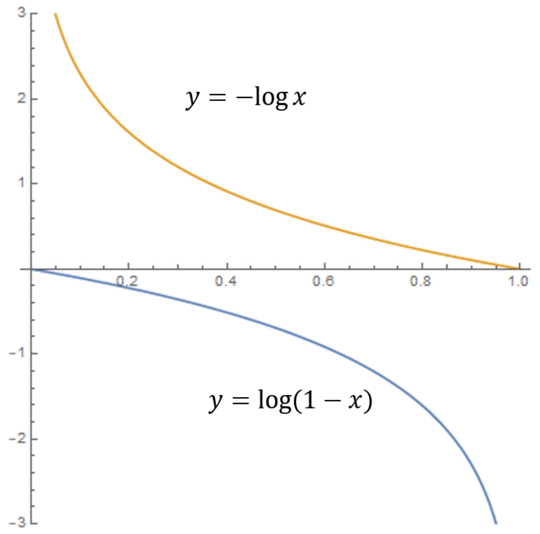
$$\min_G E_{\mathbf{z}\sim P_{\mathbf{noise}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))].$$

GAN's value function (continued)

Note that as a solution to the problem of $D(G(\mathbf{z}))$ approaching zero and the gradient for G becoming close to zero, which prevents learning from proceeding, the training of the generator is often changed to

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

, but this change in the cost function was obtained experimentally rather than theoretically [Goodfellow, §20.10.4].



Both $y = \log(1 - x)$ and $y = -\log x$ are decreasing functions.

5.7 DCGAN

Based on the GAN method, a model which uses convolutional neural networks as the generator and discriminator is called Deep Convolutional GAN (DCGAN). In DCGAN, there are some improvements such as

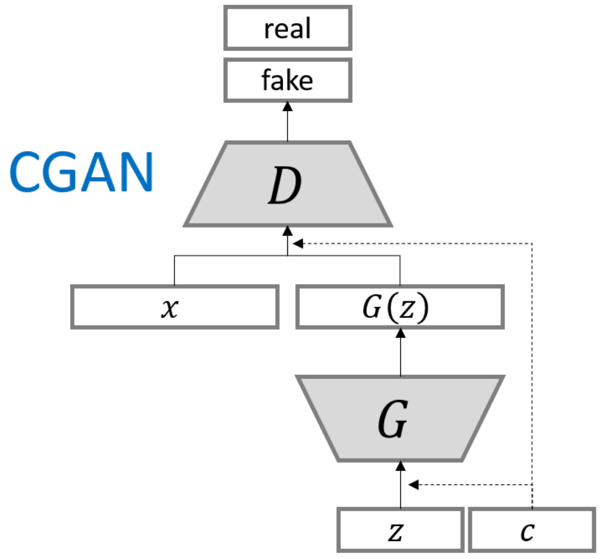
- Use a convolution layer instead of a pooling layer;
- Use tanh as the output of the generator;
- Use Leaky ReLU function defined as

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

  in the discriminator. と定義される Leaky ReLU 関数を用いる

5.8 Conditionnal GAN

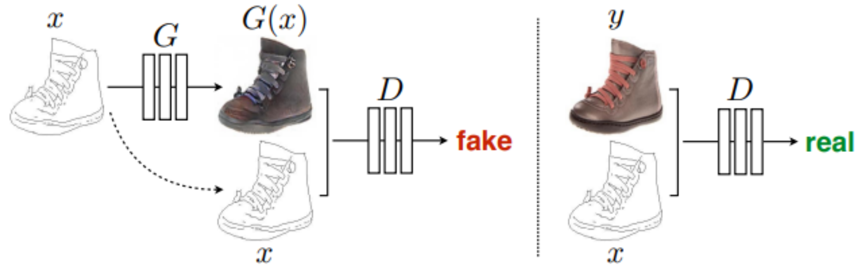
Since GAN only imitates the distribution of the real data, even if there are several classes in the training data, we cannot control which class of data is generated. In Conditional GAN, the same condition c is given to both the generator and the discriminator, so that the discriminator is trained to judge the data as real, i.e., $D(G(\mathbf{z}, c), c) > 0.5$, only when the data to be discriminated is similar to the real data with condition c (e.g., class name).



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

5.9 pix2pix

Many of you may be acquainted with pix2pix as it is often featured in the news. Pix2pix is a neural network model that performs style transformation on input images using an adversarial generative network (GAN) [Isola, P. et al., 2017]. It has been applied to automatic colorization of line drawings, style transformation of paintings (e.g., Van Gogh-style painting), conversion of gray-scale images into RGB images, and conversion of aerial photographs into map images, etc.

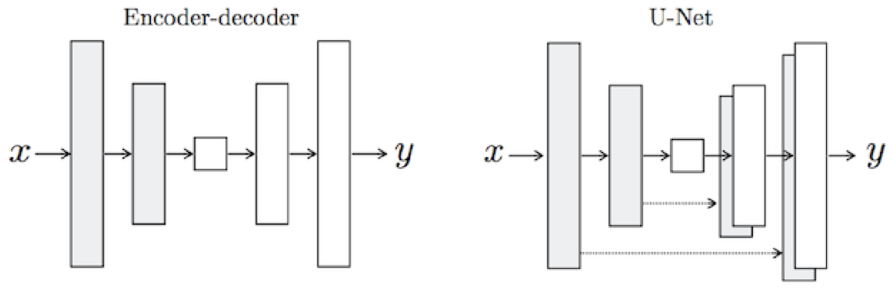


[Isola, P. et al., 2017]

When trying to colorize a line drawing, the generator tries to colorize the input line drawing. In the discriminator, a set of line drawings and a colored image are input. The input colored image can be either the one generated by the generator or the correct image. The discriminator is trained to distinguish between the two input images.

Architecture of the generator

The network of the pix2pix generator uses a network model called U-Net. U-Net is a network that uses convolution layers, deconvolution layers, and skip connections as shown in the figure. The skip connection is effective in sharing edge positions between input and output images. Note that pix2pix uses U-Net for dimensionality reduction, so it does not use noise for generation, unlike regular GANs. In addition, the training of the generator uses the error function of the GAN plus the L1 error between the correct image and the generated image.



Encoder-decoder model and U-Net [Isola et all, 2017]

Implementation example by the original proposers https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

5.10 Exercises on pix2pix

Notebook: `6-2_pix2pix-Keras.ipynb`

In this notebook, we use Keras to implement pix2pix and tackle the task of coloring a black and white image. This notebook is a rewrite of the code by Phillip Isola, one of the original proposers, for TensorFlow2.

- Anyway, let's try using pix2pix, the fruits of deep learning.
- Try other style conversion tasks if you have time.

6 General object detection—image localization, detection, and segmentation

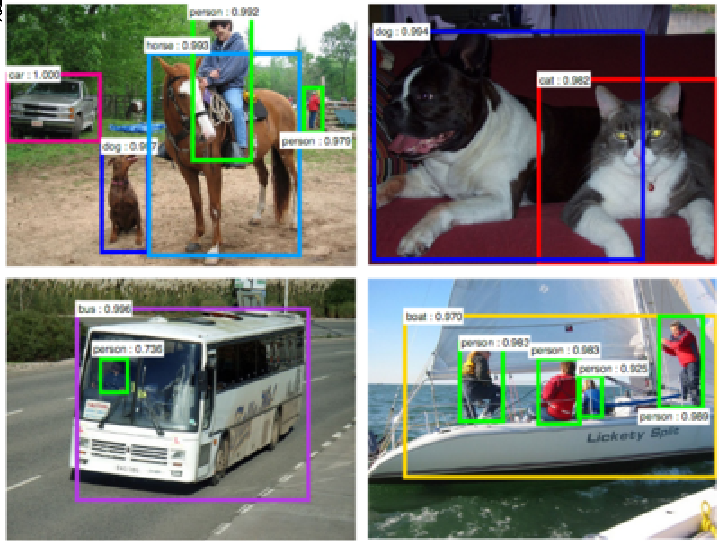
The task of detecting the position and class of a given object (person, car, product, etc.) in a given image is called general object detection. Among them, the following tasks are often used.

Classification Identify the class of objects in the image.

Localization Identify the rectangle surrounding the object of interest.

Detection Identify the class and rectangle position of the target object, if it exists.

Semantic segmentation Grasp the shape of an object by assigning to which object class each pixel in the image belongs.



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

6.1 Faster R-CNN

R-CNN

R-CNN (Regions with CNN features) is a general object detection algorithm, which works as follows.

1. Find object candidate regions (rectangular position candidates) using the existing method called Selective Search (about 2000).
2. Resize all the images of the object candidate regions to a certain size and apply CNN to extract features.
3. Use the extracted features to perform category identification by multiple SVMs (Support Vector Machines) and estimate the rectangular position more accurately by regression.

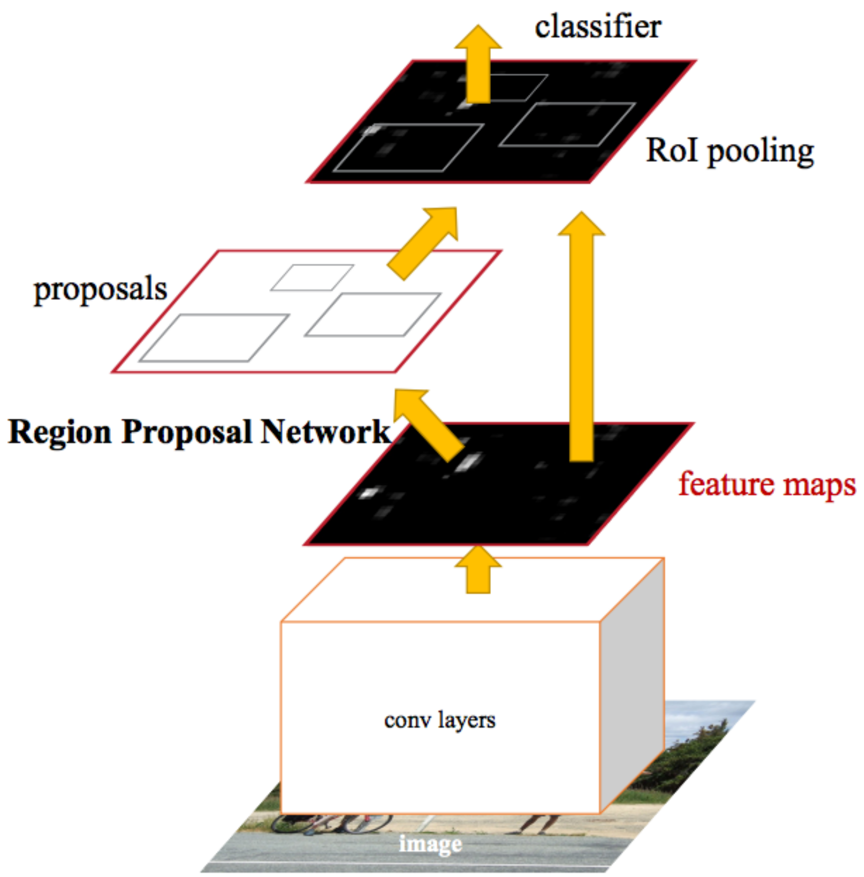
Although R-CNNs are highly accurate, they have the disadvantage that each of the above steps needs to be trained separately, and they are very time-consuming to run because CNN processing is performed for each candidate region.

Faster R-CNN

Faster R-CNN [S. Ren et al. 2015] is an algorithm that improves on R-CNN and has the following characteristics.

- Instead of Selective Search, candidate regions are proposed from the features obtained by CNN on the input image using a learnable operation (Regional Proposal Network). The candidate regions are transformed into regions of the same size by a pooling operation called RoI pooling, and the class and rectangular position are estimated respectively. This eliminates the necessity of performing a CNN for each candidate region.
- By using multi-task learning, where the cost is the sum of the costs at all stages, including the selection of candidate regions, it is possible to perform end-to-end learning (i.e., to optimize the entire model in a single learning session).

Note: The Fast R-CNN proposed just before Faster R-CNN is a model with almost the same architecture, but it uses the existing Selective Search to suggest candidate regions from features, and this part needs to be trained separately.



Faster R-CNN

https://arxiv.org/abs/1506.01497

6.2 YOLO and SSD

YOLO

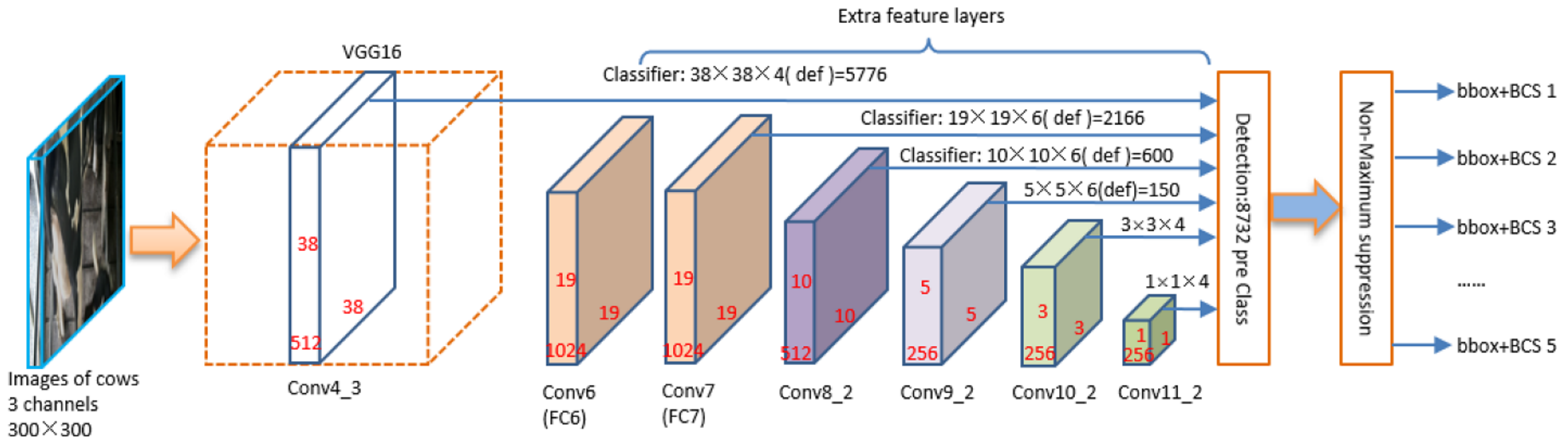
YOLO (You Only Look Once) [Redmon, J. et al., 2015] implements a fast algorithm using

1. Divide the entire image into a grid in advance;
2. Estimate one class for each cell;
3. Use it to find the rectangular region.

However, it is not suitable for processing images with many objects, since a cell can not belong to more than two rectangular areas.

SSD

SSD (Single Shot MultiBox Detector) [Liu, W. et al., 2015] is an algorithm that can recognize objects of various scales by proposing rectangular regions for feature maps in each layer of the CNN, while being faster than Faster R-CNN by using an algorithm similar to YOLO. It is also capable of end-to-end learning, and is often used as much as Faster R-CNN.



Architecture of SSD

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

Reference:

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

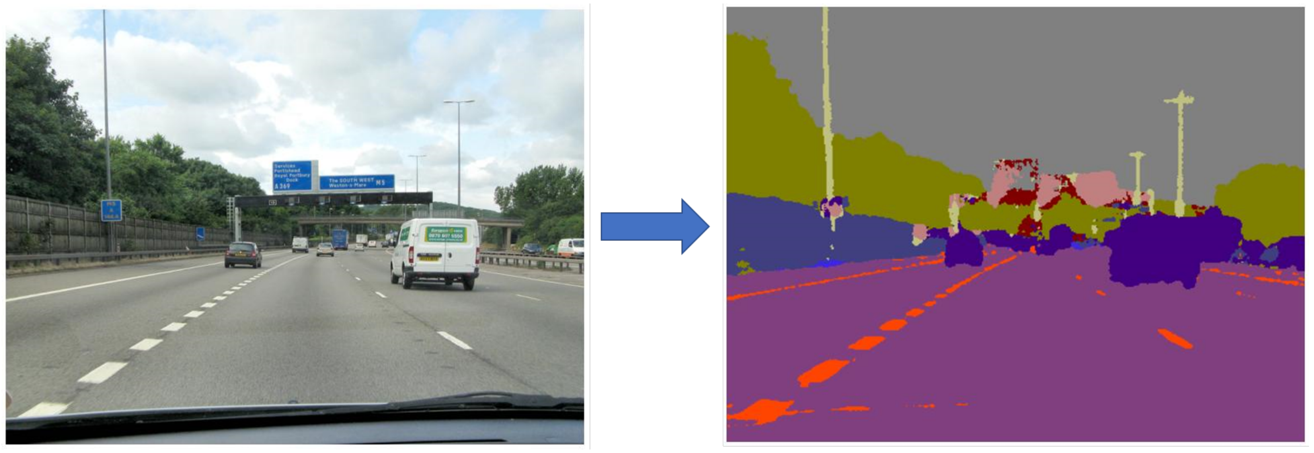
6.3 Semantic segmentation

For a given image, the task of classifying to which class each pixel in the image belongs is called semantic segmentation. This is important in the fields of autonomous vehicles and medical imaging.

The following are two typical deep learning models.

FCN Fully Convolutional Networks: similar architecture with U-Net;

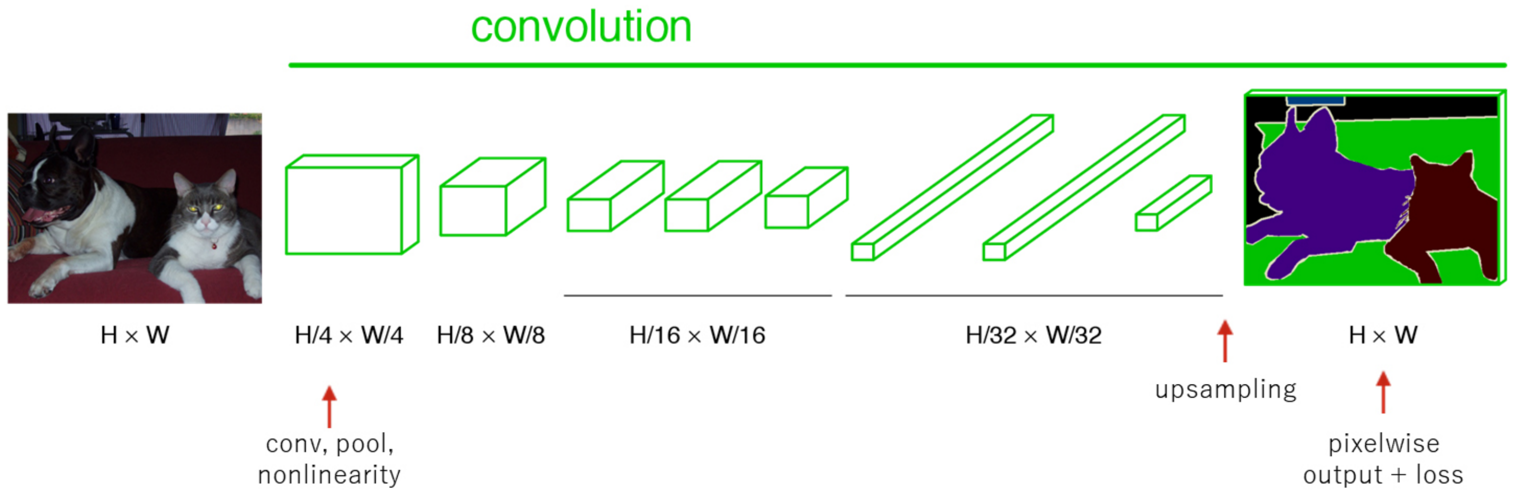
SegNet Transfer the pooling index from the encoder to the decoder.



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

FCN: Fully Convolutional Networks

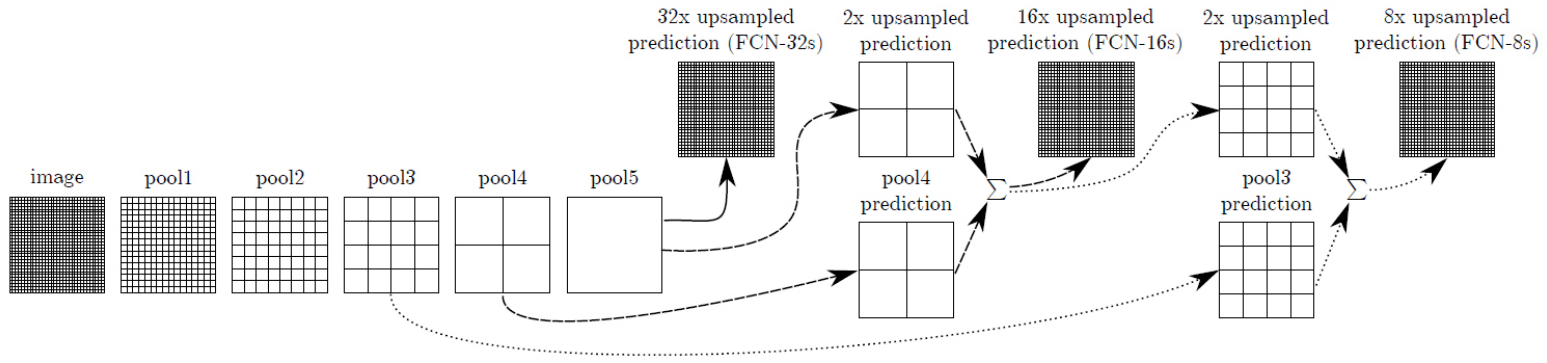
In models prior to FCN, only images of fixed size could be handled due to the presence of fully connected layers in the network, but FCN uses convolutional layers with a filter size of 1×1 instead of fully connected layers, thus eliminating the restriction of fixed image size.



Architecture of FCN https://arxiv.org/abs/1411.4038

Decoder of FCN

FCN classifies the original image pixel by pixel by classifying each pixel of the feature map obtained by applying CNN and upsampling the result using deconvolution. By using the results of each layer of the CNN as a feature map through skip connections, it can identify fine boundaries.



Upsampling of FCN https://arxiv.org/abs/1411.4038

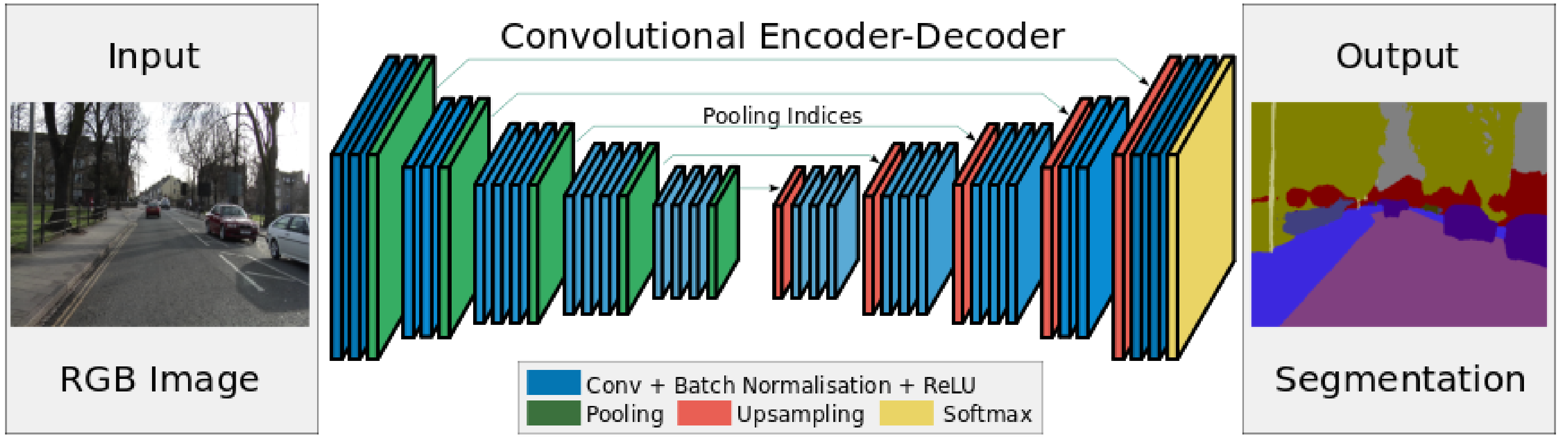
6.4 SegNet

Among the various architectures for semantic segmentation that have been proposed since FCN, the SegNet model aims to improve the efficiency of memory usage by reducing the size of the information passed from the encoder to the decoder.

Comparing to in the case of FCN, the information passed from the encoder to the decoder is as follows.

FCN Feature maps at each stage

SegNetThe final feature map and the index that took the maximum value in each max spooling layer

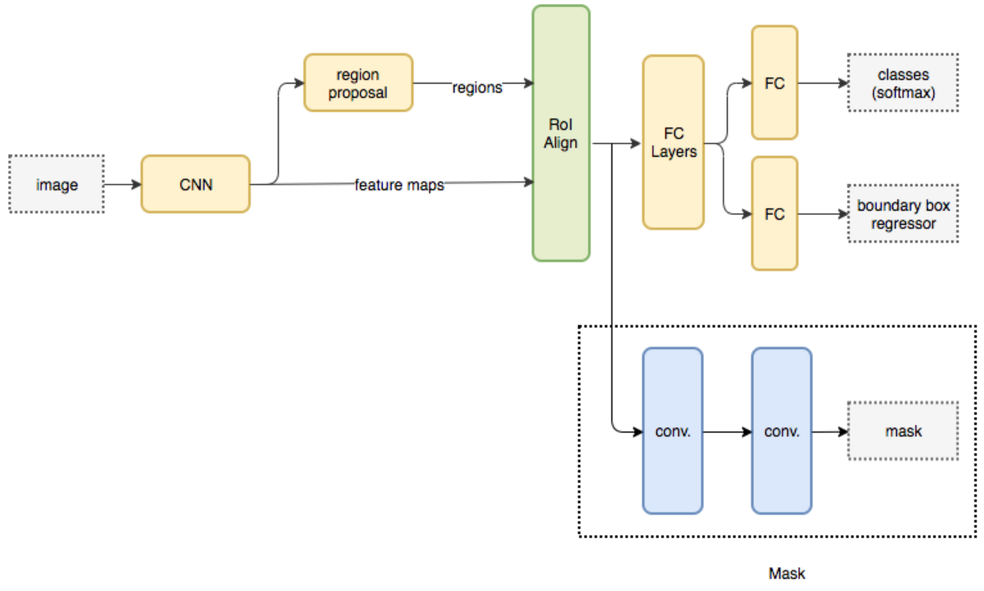


Architecture of SegNet https://arxiv.org/abs/1505.07293

6.5 Exercises on Mask R-CNN

Notebook: `3-10_MaskRCNN.ipynb`

Mask RCNN is basically a model for general object recognition that adds a semantic segmentation task to Faster RCNN, but with an improved way of proposing bounding boxes in the middle layer. [K. He, et al. 2017]. In this notebook, we try to use a library that implements Mask RCNN using Keras for video processing. The implementation we use here is provided by matterport on github.



Architecture of Mask R-CNN `https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272`