

畳み込みニューラルネットワーク

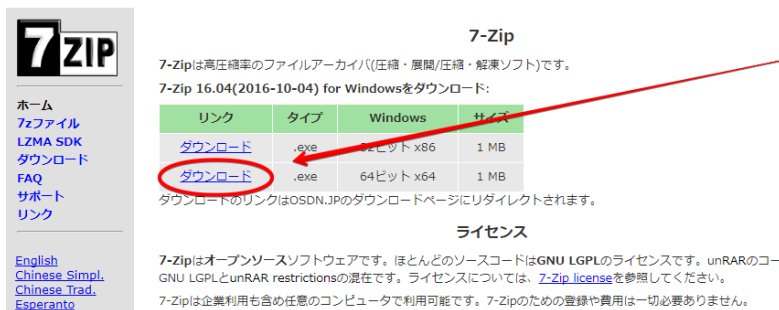
1 目的

- 畳み込みニューラルネットワークの一般的な構造を理解します。
- 畳み込みニューラルネットワークを使った画像の分類の仕組みを学びます。

2 準備

2.1 解凍用ソフト

授業では、tar ファイルを解凍するので、解凍用のソフトをインストールします。tar ファイルを解凍するソフトがインストールされていない場合には、<http://www.visihttps://sevenzip.osdn.jp/> のサイトから、「7-Zip」をダウンロードしてインストールします。



7-Zip

7-Zipは高圧縮率のファイルアーカイバ(圧縮・展開/圧縮・解凍ソフト)です。

7-Zip 16.04(2016-10-04) for Windowsをダウンロード:

リンク	タイプ	Windows	サイズ
ダウンロード	.exe	32ビット x86	1 MB
ダウンロード	.exe	64ビット x64	1 MB

ダウンロードのリンクはOSDN.JPのダウンロードページにリダイレクトされます。

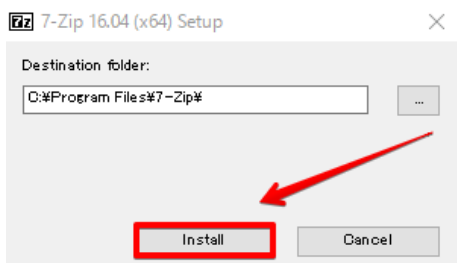
ライセンス

7-Zipはオープンソースソフトウェアです。ほとんどのソースコードはGNU LGPLのライセンスです。unRARのコードは、GNU LGPLとunRAR restrictionsの混在です。ライセンスについては、[7-Zip license](#)を参照してください。

7-Zipは企業利用も含め任意のコンピュータで利用可能です。7-Zipのための登録や費用は一切必要ありません。

クリックしてダウンロード

ダウンロードしたファイルをダブルクリックして実行し、インストールを行います。



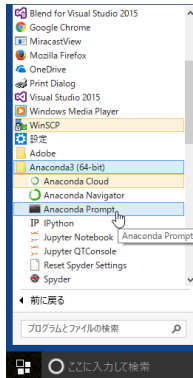
7-Zip 16.04 (x64) Setup

Destination folder:
C:\Program Files\7-Zip\

Install Cancel

クリック

2.2 実行環境の選択



Anaconda Prompt を選択します。

chenv に入ります。

```
activate chenv
```

2.3 授業関連ファイルの取得

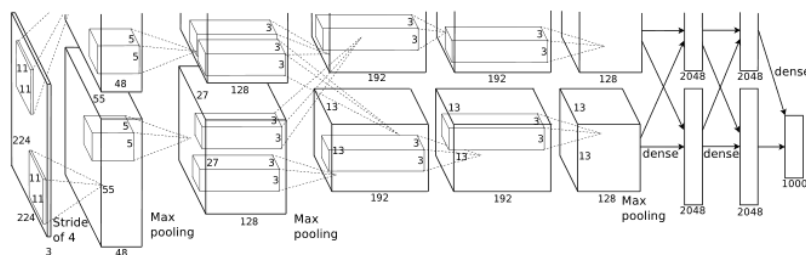
授業の関連ファイルをダウンロードします。

```
cd Lesson
git clone https://github.com/k1nk/ch3.git
```

ch3 フォルダが作成されます。その下に関連ファイルがあります。

3 Alexnet の構造

3.1 Alexnet の構造



AlexNet の構造

alex.py

```
1 import numpy as np
2
3 import chainer
4 import chainer.functions as F
```

```

5 from chainer import initializers
6 import chainer.links as L
7
8
9 class Alex(chainer.Chain):
10
11     """Single-GPU AlexNet without partition toward the channel axis."""
12
13     insize = 227
14
15     def __init__(self):
16         super(Alex, self).__init__()
17         with self.init_scope():
18             self.conv1 = L.Convolution2D(None, 96, 11, stride=4)
19             self.conv2 = L.Convolution2D(None, 256, 5, pad=2)
20             self.conv3 = L.Convolution2D(None, 384, 3, pad=1)
21             self.conv4 = L.Convolution2D(None, 384, 3, pad=1)
22             self.conv5 = L.Convolution2D(None, 256, 3, pad=1)
23             self.fc6 = L.Linear(None, 4096)
24             self.fc7 = L.Linear(None, 4096)
25             self.fc8 = L.Linear(None, 1000)
26
27     def __call__(self, x, t):
28         h = F.max_pooling_2d(F.local_response_normalization(
29             F.relu(self.conv1(x))), 3, stride=2)
30         h = F.max_pooling_2d(F.local_response_normalization(
31             F.relu(self.conv2(h))), 3, stride=2)
32         h = F.relu(self.conv3(h))
33         h = F.relu(self.conv4(h))
34         h = F.max_pooling_2d(F.relu(self.conv5(h)), 3, stride=2)
35         h = F.dropout(F.relu(self.fc6(h)))
36         h = F.dropout(F.relu(self.fc7(h)))
37         h = self.fc8(h)
38
39         loss = F.softmax_cross_entropy(h, t)
40         chainer.report({'loss': loss, 'accuracy': F.accuracy(h, t)},
41                        self)
42         return loss

```

python でクラスを定義する際に、`__init__` と、 `__call__` というメソッドを作成した場合、まず、

`a = Alex()`

として、インスタンスを生成した場合に、`__init__`メソッドが呼び出されます。

そして、作成したインスタンスを、

`loss = a(x,t)`

として呼び出した時に、`__call__`メソッドが呼び出されます。

`loss = a(x,t)`

として作成したインスタンスを呼び出した時、入力された画像 `x` は、`__call__`メソッドにおいて、

`self.conv1 → F.relu → F.local_response_normalization → F.max_pooling_2d`

として処理されます。このパターンの処理が、`F.local_response_normalization` や `F.max_pooling_2d` が省略される場合もありますが、何回か続きます。これにより、画像の特徴をいくつかのチャンネルにわけて抽出します。そして、抽出した特徴をもとに全結合層を何回か使って、具体的な分類を行います。

3.1.1 入力データ (chainer.links.Convolution2D)

チャンネル (入力データは、赤緑青の3つのチャンネルを持ちます) および縦、横のサイズがそれぞれ、(3、2 2 7、2 2 7) のデータです。学習を行う際には、複数のデータをバッチとして入力します。したがって、渡すデータは (バッチの数, チャンネルの数, 縦サイズ, 横サイズ) という配列になります。

3.1.2 畳み込み層 (chainer.links.Convolution2D)

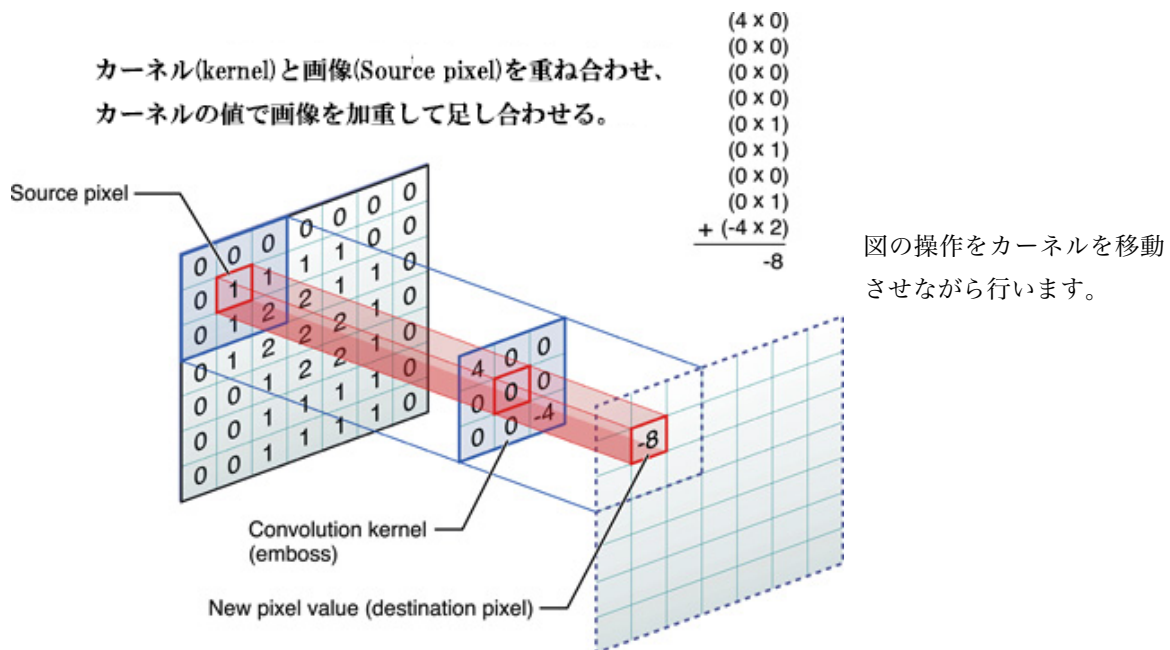
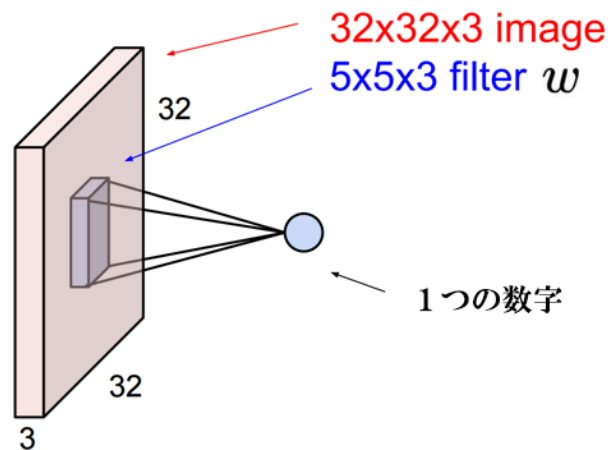


図1 畳み込みの計算

出典:<https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html>

通常は、カーネルのサイズの関係で、出力画像のサイズは、入力画像よりも小さくなります。入力画像の周辺を0などの数値で埋めてから操作を行うことにより、出力画像の大きさを入力画像の大きさと同じにすることもあります。これをパディングといいます。

ディープラーニングでは、このカーネルのウェイトを学習します。



入力画像がRGBのような複数チャンネルからなっている場合、カーネル(フィルタ)も画像と同じ複数のチャンネルからなります。

図2 畳み込みの計算



1つのカーネル(フィルタ)から、1つのチャンネル(特徴マップ)が出力されます。

図3 カーネルとチャンネル

出典：http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf

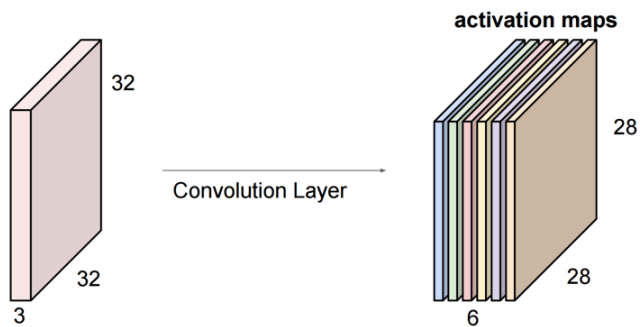
```
18 self.conv1 = L.Convolution2D(None, 96, 11, stride=4)
19 self.conv2 = L.Convolution2D(None, 256, 5, pad=2)
```

の各パラメータは、

L.Convolution2D(入力チャンネルの数, 出力チャンネルの数, カーネルの大きさ, stride=ストライド)

を表しています。

入力チャンネルの数は、None となっています。None を指定した場合、最初にデータが与えられたときに、入



カーネルの数だけ、チャンネルが出力されます。

図4 畳み込み層の計算方法



学習したフィルタの例です。この例では、 $1 \times 1 \times 1$ のフィルタ（96チャンネル）となっています。層の方向のデータは色で表されています。

図5 カーネル（フィルタ）の例

入力データから自動的に入力チャンネルの数が決まります。

出力チャンネル数は96となります。カーネルの数も同じだけあることとなります。

stride は、カーネルを動かすときの幅の大きさです。指定しない場合には1となります。

pad は、入力画像をパディングする場合の大きさです。

縦、横 10×10 の3チャンネルのデータを作成して、挙動を確認してみましょう。

```
activate chenv
python
>>> import numpy as np
>>> import chainer.links as L
>>> x = np.arange(-1 * 3 * 10 * 10 / 2, 1 * 3 * 10 * 10 / 2, dtype='f').
    reshape(1, 3, 10, 10)
>>> x
array([[[[-150., -149., -148., -147., -146., -145., -144., -143.,
          -142., -141.], ...,
        [ 140.,  141.,  142.,  143.,  144.,  145.,  146.,  147.,  148.,
          149.]]]], dtype=float32)
```

```
>>> l = L.Convolution2D(3, 7, 5)
>>> y = l(x)
>>> y.shape
(1, 7, 6, 6)
```

入力チャンネル数が3、出力チャンネル数が7、カーネルの大きさは 5×5 の Convolution2D の操作をおこなっています。出力画像は7チャンネル、画像のサイズは 6×6 となります。

`chainer.links.Convolution2D` <https://docs.chainer.org/en/stable/reference/generated/chainer.links.Convolution2D.html>

3.1.3 ReLU (`chainer.functions.relu`)

ReLU (Rectified Linear Unit) は活性化関数の一種です。

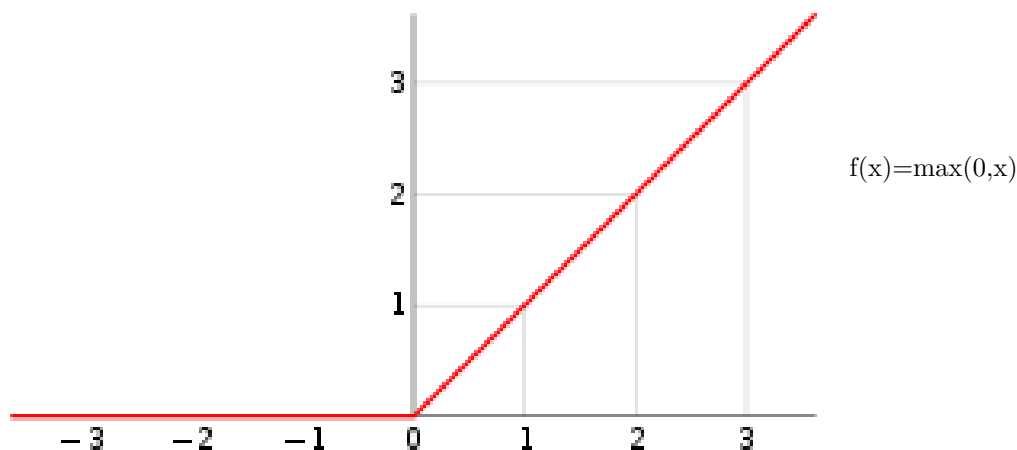


図6 ReLU

$f(x)=\max(0,x)$ は、

$$f(x) = \begin{cases} x & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$

と表すこともできます。活性化関数の前の層の計算結果がプラスになる場合には、信号をそのまま通過させ、そうでない場合には、信号を通過させずに0にする機能をもつと言うこともできます。線形結合から成る層の後ろに、このような一種の条件分岐の機能を持つ関数を配置することにより、単なる入力信号の線形結合のみではできない機能をネットワークに与えることが可能となります。

また、重みの勾配を計算する際にも、ReLUの傾きが0となっている部分の重みは更新されず、ReLUを信号が通過して最終的な損失に影響を及ぼす部分の重みが更新されることとなります。そういう意味において、ReLUは更新される重みの範囲にも影響を与え、ネットワーク全体の機能の生成に一定の役割を果たしていると思われます。

ReLU を活性化関数として用いると、他の活性化関数 ($f(x) = \tanh(x)$ など) を使った場合よりも、訓練時の誤差が一定になるまでの学習回数が少なくて済むことが報告されています。

3.1.4 シグモイド関数

活性化関数には、ReLU 以外の非線形な関数が用いられることがあります。線形な関数の後ろに非線形な関数を配置することにより、線形結合のみでは不可な機能や表現をネットワークに与えます。ReLU 以外の活性化関数の例として、代表的なものにシグモイド関数があります。シグモイド関数の特徴は、関数の範囲が0か

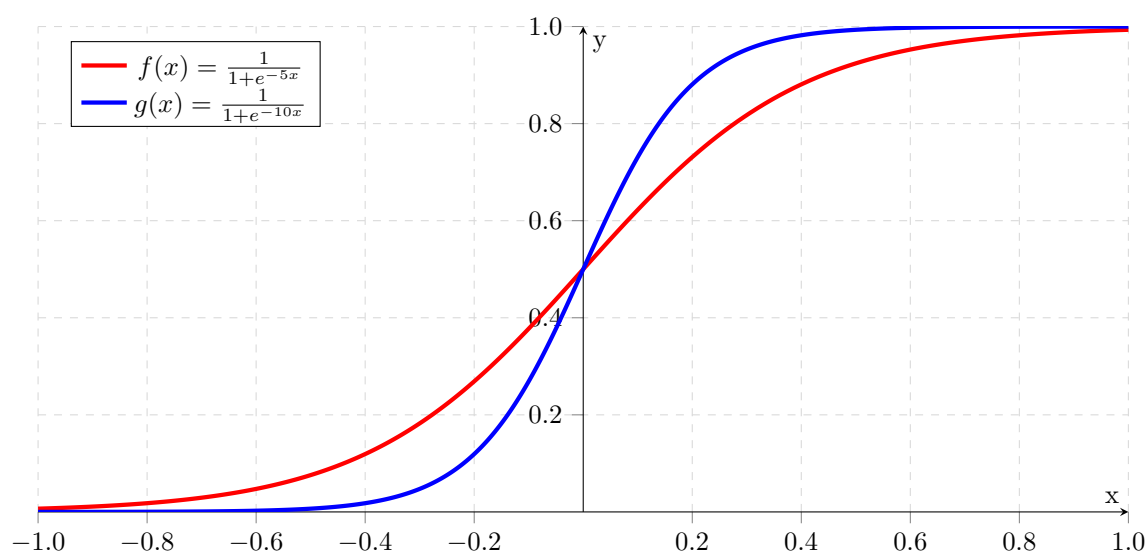


図7 シグモイド関数

ら1までということです。また、グラフをみると判りますが、入力される値が非常に大きい場合、あるいは、非常に小さい場合には、シグモイド関数の傾きは非常に小さくなります。したがって、シグモイド関数の前の層から出力される値がそのような場合には、前の層の値が変化しても、シグモイド関数から出力される値の変化は非常に小さくなります。

ReLU 関数の場合には、前の層から出力される値がプラスであれば、後の層に信号の変化がそのまま伝達されるのに対して、シグモイド関数の場合には、信号の変化を有効に後の層へ伝えることができる入力信号の値の範囲が ReLU よりも狭いと言えます。

また、ReLU 関数の場合には、信号が伝わる場合には、その傾きが1であるため、信号の変化がそのまま後の層に伝わります。標準的なシグモイド関数の場合、その傾きの最大値は0.25です。したがって、前の層の出力が1変化しても、後の層の信号の変化が最大でもその4分の1に弱められます。そのため、シグモイド関数をつかって複数の階層をつなげていくと、入力層に近い層のウェイトを変化させても、出力層へ出力される値の変化がほとんどなくなってしまいます。これを勾配の消失といいます。

このような問題があるため、シグモイド関数は中間層でなく、出力層の活性化関数として、最終的に出力される値を0から1の範囲に調整するために用いられます。

```
>>> import chainer.functions as F
>>> F.relu(x)
```


x の負の値の部分が0 となっていることを確認してみましょう。

3.1.15 局所的応答正規化、Local Response Normalization(LRN)

chainer.functions.local_response_normalization

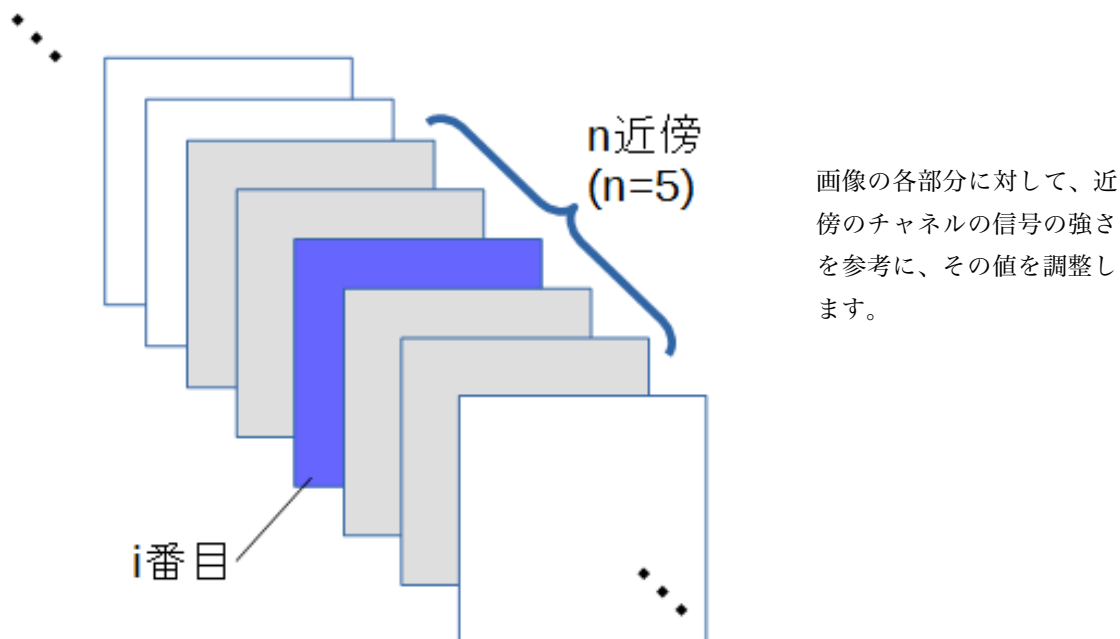


図8 局所的応答正規化

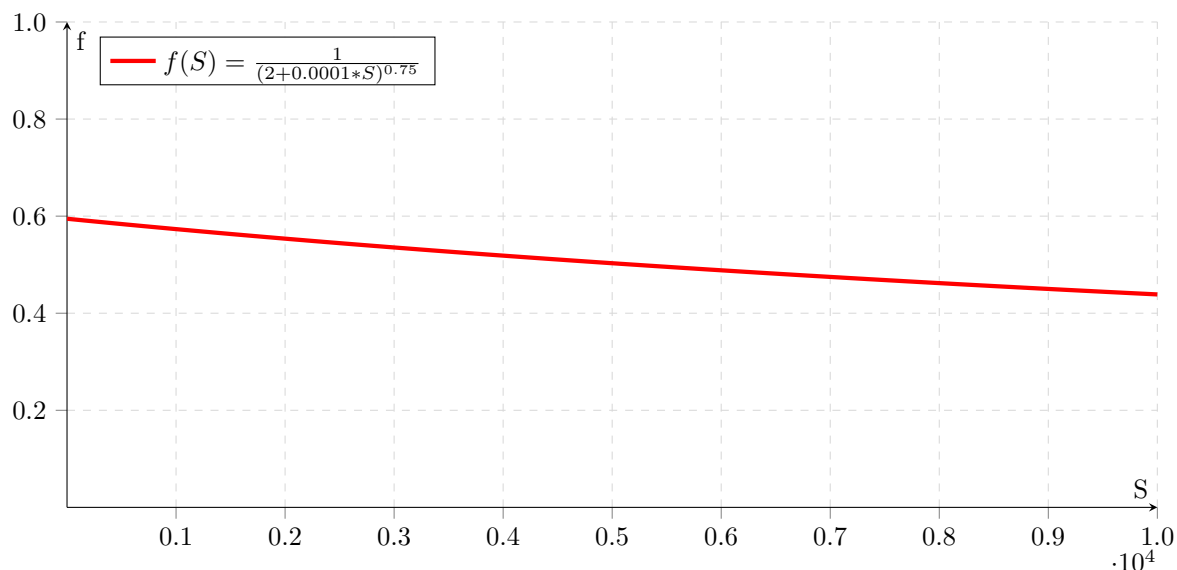
$$b_{x,y}^i = a_{x,y}^i / \left(k + \alpha \sum_{j=\max(0, i-\frac{n}{2})}^{\min(N-1, i+\frac{n}{2})} (a_{x,y}^j)^2 \right)^\beta$$

$a_{x,y}^i$ は i 番目のチャンネル (特徴マップ) の (x,y) のピクセルを、N はチャンネルの総数を表します。k, n, α , β が固定的なパラメータを表します。例: k = 2, n = 5, $\alpha = 10^{-4}$, $\beta = 0.75$

図の出典は: <http://may46onez.hatenablog.com/entry/2016/01/08/142843>

$$S = \sum_{j=\max(0, i-\frac{n}{2})}^{\min(N-1, i+\frac{n}{2})} (a_{x,y}^j)^2$$

とすると、S は $a_{x,y}^i$ のチャンネル方向の n 程度の近傍の信号の強さの合計を表します。これに係数 $f(S)=1/(k+\alpha S)^\beta$ を乗じて修正を行います。k = 2, $\alpha = 10^{-4}$, $\beta = 0.75$ の場合、この係数は図のような大きさになります。



これは、画像の同じ場所のチャネル方向の n 程度の近傍に信号の強い部分がある場合、そうでない場合と比べて信号がより弱められることを示しています。最適化の学習過程において、近傍に複数の強い特徴がある場合、お互いを弱めあう効果を入れることにより一種の生存競争をおこさせることを狙っているのではと思われます。このような効果を側方抑制効果といいます。

3.1.6 最大プーリング (chainer.functions.max_pooling_2d)

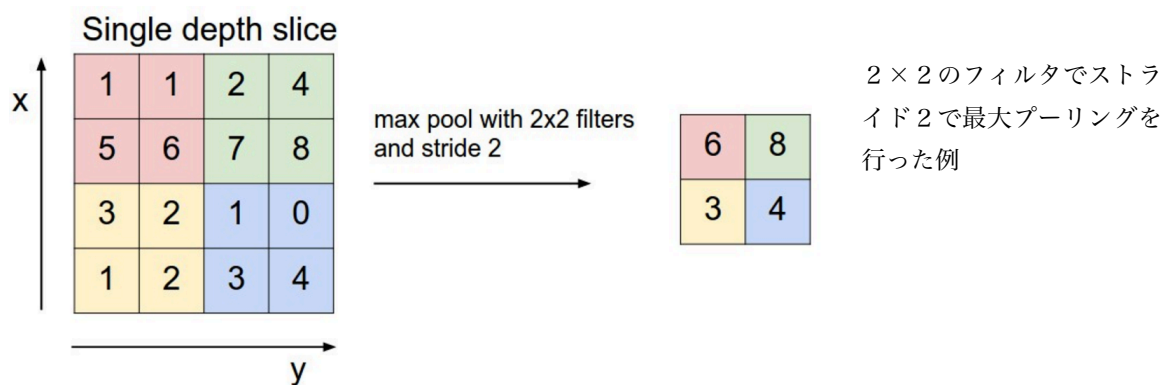


図9 最大プーリングの計算例

出典:<http://cs231n.github.io/convolutional-networks/#pool>

Convolution2D では、画像の一定の範囲に対してカーネルの値を乗じて出力を計算していました。このかわりに、カーネルを使うのではなく、一定の範囲（フィルタ）の最大値をとる動作をおこなうのが、最大プーリングです。この操作はチャネルごとに別々に行われます。したがって、入力チャネルの数と出力チャネルの数は変わりません。最大プーリングを行うことにより、画像上にある特徴の位置が多少変化しても、同じよう

な反応をえることができようになります。

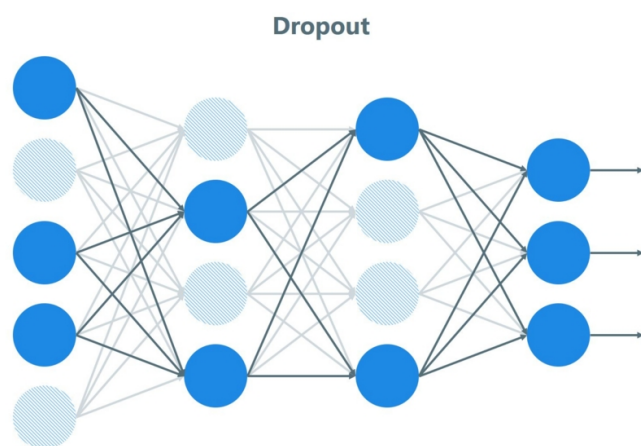
```
chainer.functions.max_pooling_2d
```

https://docs.chainer.org/en/stable/reference/generated/chainer.functions.max_pooling_2d.html#chainer.functions.max_pooling_2d

```
>>> mx = F.max_pooling_2d(x,3, stride=1) フィルタの大きさが3×3 ストライドが1#
>>> mx.shape
(1L, 3L, 8L, 8L)
>>> mx
```

3.1.7 ドロップアウト Dropout (chainer.functions.dropout)

学習中に一定の確率 $ratio$ でランダムに要素を振るい落とします。そして残った要素を $\frac{1}{ratio}$ 倍します。学習中でない場合には、単に入力された値をそのまま返します。



学習を行う際に、ノードのいくつかを無効にして学習を行います。

図 10 ドロップアウト

出典:<http://sonickun.hatenablog.com/entry/2016/07/18/191656>

ドロップアウトを行うことにより、学習時にネットワークの自由度を強制的に小さくして汎化性能を上げ、過学習を避けることができます。アンサンブル学習とは個々に学習した識別器を複数用意し、出力の平均を取るなど、それらをまとめあげて一つの識別器とする方法をいいます。ドロップアウトが効果があるのは、この「アンサンブル学習」の近似になるからであるとも言われています。

実際に、ドロップアウトの挙動を確かめてみましょう。

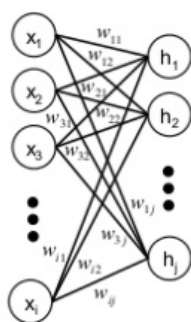
```
activate chenv
python
>>> import numpy as np
>>> import chainer
>>> import chainer.functions as F
```

```
>>> x = np.arange(-1 * 3 * 10 * 10 / 2, 1 * 3 * 10 * 10 / 2, dtype='f').
      reshape(1, 3, 10, 10)
>>> x
array([[[[-150., -149., -148., -147., -146., -145., -144., -143.,
          -142., -141.], ...,
[ 140.,  141.,  142.,  143.,  144.,  145.,  146.,  147.,  148.,
          149.]]]], dtype=float32)
>>> F.dropout(x)
variable([[[[ -0.,   -0., -296.,   -0., -292., -290.,   -0., -286.,
          -0.,   -282.], ...,
[ 280.,  282.,  284.,    0.,    0.,  290.,  292.,    0.,    0.,
          298.]]]])
>>> with chainer.using_config('train', False):
...     y = F.dropout(x) 前にインデントのスペースをいれる#
array([[[[-150., -149., -148., -147., -146., -145., -144., -143.,
          -142., -141.], ...,
[ 140.,  141.,  142.,  143.,  144.,  145.,  146.,  147.,  148.,
          149.]]]], dtype=float32)
```

with chainer.using_config('train', False): としてから F.dropout を呼ぶと、学習中でない状態で、関数を呼び出すことができます。

chainer.functions.dropout <https://docs.chainer.org/en/stable/reference/generated/chainer.functions.dropout.html>

3.1.8 全結合層 (chainer.links.Linear)



$$h_j = f(\underline{W}^T x + b_j)$$

各ノードとの結合重み

例えば、
 h_1 は $w_{11}x_1 + w_{12}x_2 + \dots + w_{1i}x_i + b_1$
 を算出し、activation functionに与えて値を得る

全てのノードがウェイトを介してつながっています。

図 11 全結合層

出典:<https://www.slideshare.net/Takayosi/nvidia-51814334>

全結合層の入力のすべての要素（ノード）が出力のすべての要素（ノード）にウェイト W を通じて繋がっ

ているため、全結合層 (fully-connected layer) とも言います。全結合層は、 $h = Wx + b$ というレイヤーを表します。ここで W はウェイトを b はバイアスを表します。 x が入力ベクトル、 h が出力ベクトルです。

chainer では、

```
model = chainer.links.Linear(<入力ベクトルのサイズ>, <出力ベクトルのサイズ>)
```

としてモデルを作成します。

入力データは縦横の画像が複数のチャンネルに積み重なっています。chainer.links.Linear にこのようなデータを入力すると、1 列の入力データに自動的に変換した上で、計算を行ってくれます。

3.1.9 ソフトマックス関数 (chainer.functions.softmax)

ソフトマックス関数は、出力層の値を確率として扱える値に変換する関数です。

$$y_i = \frac{\exp(a_i)}{\sum_j^D \exp(a_j)}$$

特徴は、 $0 \leq y_i \leq 1$ という範囲におさまり、 $\sum_i y_i = 1$ であるため、確率として扱うことができるということです。

ソフトマックス関数は、 y_i の値のなかで、より大きい値を指数関数により強調した上で、確率として扱える値に変換します。例えば、 $[10, 2, 1]$ という値はソフトマックス関数により、 $[0.999541338, 0.000335309, 0.000123353]$ となり、最も大きい値が変換後は 1 に近くなり、他の値は 0 に近くなります。

chainer.functions.softmax

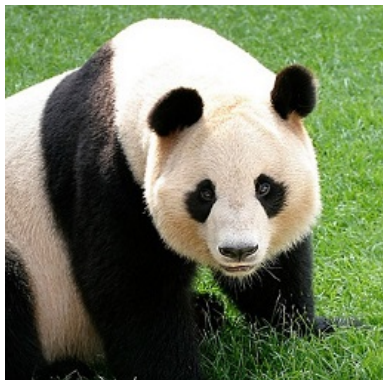
<https://docs.chainer.org/en/stable/reference/generated/chainer.functions.softmax.html>

4 学習済みデータを使った予測

事前に訓練した学習済みモデル (result_alex/model_iter_15000) を用いて予測を行います。学習済みデータは、github のファイル上限 100M byte を超えるため、USB で配布します。result_alex の下にコピーして下さい。

what_is_this_1.jpg は、 256×256 にクロップされた画像ファイルです。このファイルを認識させてみましょう。

```
python inspection2.py what_is_this_1.jpg
```



what_is_this_1.jpg

```
Anaconda Prompt
(chenv) C:\Users\nakatanikenichi\Lesson\chainer_imagenet_tools>python inspection
2.py what_is_this_1.jpg
#1 panda | 100.0%
#2 dalmatian | 0.0%
#3 soccer_ball | 0.0%
#4 metronome | 0.0%
#5 yin_yang | 0.0%
#6 rooster | 0.0%
#7 camera | 0.0%
#8 BACKGROUND Google | 0.0%
#9 sea_horse | 0.0%
#10 inline_skate | 0.0%
#11 wheelchair | 0.0%
#12 anchor | 0.0%
#13 snoopy | 0.0%
#14 gramophone | 0.0%
#15 pagoda | 0.0%
#16 headphone | 0.0%
#17 buddha | 0.0%
#18 windsor_chair | 0.0%
#19 dragonfly | 0.0%
#20 binocular | 0.0%
(chenv) C:\Users\nakatanikenichi\Lesson\chainer_imagenet_tools>
```

```
python resize.py <filename1> <filename2> ...
```

とすると、jpg のファイルを 256×256 にクロップします。クロップされたファイルは、resized フォルダの下に保存されます。自分で任意のファイルを認識させて、101 のカテゴリーのどれに分類されるか試してみしましょう。

```
python resize.py <filename.jpg>
python inspection2.py resized/<filename.jpg>
```

inspection2.py

```
1 from __future__ import print_function
2 import argparse
3 import datetime
4 import json
5 import multiprocessing
6 import random
7 import sys
8 import threading
9 import time
10
11 import numpy as np
12 from PIL import Image
13
14
15 import six
16 #import six.moves.cPickle as pickle
17 import cPickle as pickle
18 from six.moves import queue
19
```

```

20 import chainer
21 import matplotlib.pyplot as plt
22 import numpy as np
23 import math
24 import chainer.functions as F
25 import chainer.links as L
26 from chainer.links import caffe
27 from matplotlib.ticker import *
28 from chainer import serializers
29
30 parser = argparse.ArgumentParser(
31     description='Image inspection using chainer')
32 parser.add_argument('image', help='Path to inspection image file')
33 parser.add_argument('--model', '-m', default='model', help='Path to model
34     file')
35 parser.add_argument('--mean', default='mean.npy',
36     help='Path to the mean file (computed by
37         compute_mean.py)')
38
39 args = parser.parse_args()
40
41
42 def read_image(path, center=False, flip=False):
43     image = np.asarray(Image.open(path)).transpose(2, 0, 1)
44     if center:
45         top = left = cropwidth / 2
46     else:
47         top = random.randint(0, cropwidth - 1)
48         left = random.randint(0, cropwidth - 1)
49     bottom = model.insize + top
50     right = model.insize + left
51     image = image[:, top:bottom, left:right].astype(np.float32)
52     image -= mean_image[:, top:bottom, left:right]
53     image /= 255
54     if flip and random.randint(0, 1) == 0:
55         return image[:, :, ::-1]
56     else:
57         return image
58
59 import nin
60 import alex

```

```

58
59 #mean_image = pickle.load(open(args.mean, 'rb'))
60 mean_image = np.load(args.mean)
61
62 #model = nin.NIN()
63 model = alex.Alex()
64
65 #serializers.load_hdf5("gpu1out.h5", model)
66 #serializers.load_hdf5("cpu1out.h5", model)
67 serializers.load_npz("result_alex/model_iter_15000", model)
68 cropwidth = 256 - model.insize
69 model.to_cpu()
70
71 def predict(net, x):
72     h = F.max_pooling_2d(F.relu(net.mlpconv1(x)), 3, stride=2)
73     h = F.max_pooling_2d(F.relu(net.mlpconv2(h)), 3, stride=2)
74     h = F.max_pooling_2d(F.relu(net.mlpconv3(h)), 3, stride=2)
75     #h = net.mlpconv4(F.dropout(h, train=net.train))
76     h = net.mlpconv4(F.dropout(h))
77     h = F.reshape(F.average_pooling_2d(h, 6), (x.data.shape[0], 1000))
78     return F.softmax(h)
79
80 def predict_alex(net,x):
81     h = F.max_pooling_2d(F.local_response_normalization(
82         F.relu(net.conv1(x))), 3, stride=2)
83     h = F.max_pooling_2d(F.local_response_normalization(
84         F.relu(net.conv2(h))), 3, stride=2)
85     h = F.relu(net.conv3(h))
86     h = F.relu(net.conv4(h))
87     h = F.max_pooling_2d(F.relu(net.conv5(h)), 3, stride=2)
88     h = F.dropout(F.relu(net.fc6(h)))
89     h = F.dropout(F.relu(net.fc7(h)))
90     h = net.fc8(h)
91
92     return F.softmax(h)
93
94 #setattr(model, 'predict', predict)
95
96 img = read_image(args.image)
97 x = np.ndarray(

```



```

98         (1, 3, model.insize, model.insize), dtype=np.float32)
99 x[0]=img
100 #x = chainer.Variable(np.asarray(x), volatile='on')
101 x = chainer.Variable(np.asarray(x))
102 with chainer.using_config('train', False):
103     with chainer.no_backprop_mode():
104         #score = predict(model,x)
105         score = predict_alex(model,x)
106         #score=cuda.to_cpu(score.data)
107
108 categories = np.loadtxt("labels.txt", str, delimiter="\t")
109
110 top_k = 20
111 prediction = zip(score.data[0].tolist(), categories)
112 prediction.sort(cmp=lambda x, y: cmp(x[0], y[0]), reverse=True)
113 for rank, (score, name) in enumerate(prediction[:top_k], start=1):
114     print('%#d | %s | %4.1f%%' % (rank, name, score * 100))

```

5 画像データのダウンロード

学習済みモデル作成の元となった画像データをダウンロードして解凍します。http://www.vision.caltech.edu/Image_Datasets/Caltech101/ のサイトから、「101_ObjectCategories.tar.gz」をダウンロードします。

Caltech 101

 [Caltech256](#) 

[\[Description\]](#) [\[Download\]](#) [\[Discussion\]](#) [\[Other Datasets\]](#)



Description

Pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels. We have carefully clicked outlines of each object in these pictures, these are included under the 'Annotations.tar'. There is also a matlab script to view the annotations, 'show_annotations.m'.

How to use the dataset

If you are using the Caltech 101 dataset for testing your recognition algorithm you should try and make your results comparable to the results of others. We suggest training and testing on fixed number of pictures and repeating the experiment with different random selections of pictures in order to obtain error bars. Popular number of training images: 1, 3, 5, 10, 15, 20, 30. Popular numbers of testing images: 20, 30. See also the discussion below.

When you report your results please keep track of which images you used and which were misclassified. We will soon publish a more detailed experimental protocol that allows you to report those details. See the Discussion section for more details.

Download

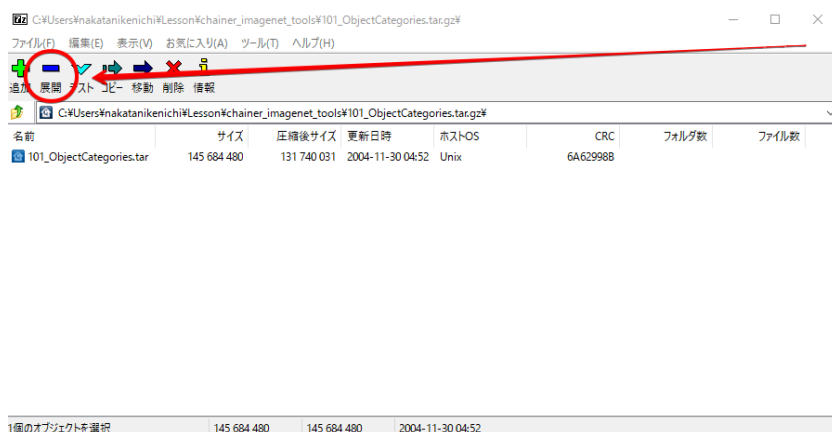
Collection of pictures: [101_ObjectCategories.tar.gz \(131Mbytes\)](#)

Outlines of the objects in the pictures: [1] [Annotations.tar](#) [2] [show_annotation.m](#)

クリックしてダウンロード

ファイルを解凍します。

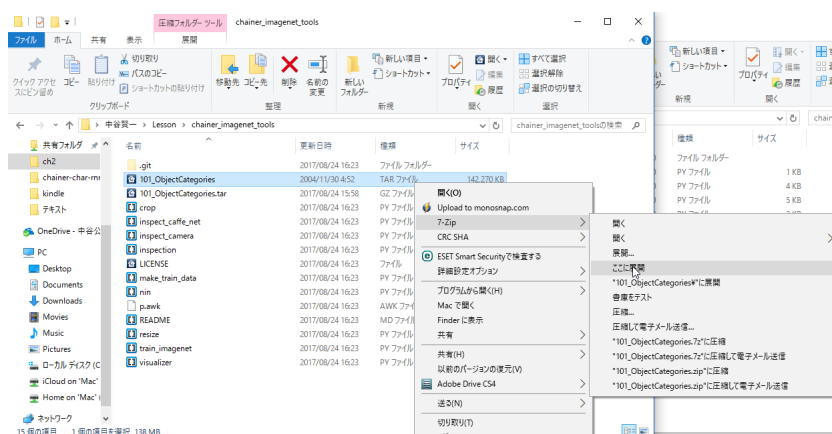
「101_ObjectCategories.tar.gz」を解凍すると、「101_ObjectCategories.tar」が生成されます。それを再度、解凍すると、「101_ObjectCategories」フォルダが作成されます。



クリックして展開



クリック



tar ファイルを右クリックして、「ここに展開」

「101_ObjectCategories」フォルダが作成されます。「101_ObjectCategories」フォルダには、「airplanes」「camera」など、101のカテゴリごとのフォルダがあります。それぞれのフォルダの中には、そのカテゴリに属するものの写真が保存されています。内容を確認してみましょう。

6 (オプション) Chainer サンプルのダウンロード

chainer のソースをダウンロードします。chainer のソースの中にサンプルコードがあります。

```
git clone https://github.com/chainer/chainer.git
```

/examples/imagenet の下にコードがありますのでエディタ等で内容を確認して見ましょう。それ以外のサンプルも確認してみましょう。