

農学生命情報科学特論 I

ICT や IoT 等の先端技術を活用し、効率よく高品質生産を可能にするスマート農業への取り組みは世界的に進められています。その基礎を支えている技術の一つがプログラミング言語。なかでも、習得しやすくかつ応用範囲の広い Python がとくに注目されています。本科目では、農学や分子生物学などの分野で利用される Python の最新事例を紹介しながら、Python の基礎文法の講義を行います。

孫 建強 <https://aabbdd.jp/>

農研機構・農業情報研究センター

June

08

17:15–20:30

基本構文

第 1 回目の授業では、プログラミング言語の基本であるデータ構造とアルゴリズムを簡単に紹介してから、Python の基本構文を紹介する。Python のスカラー、リスト、ディクショナリ、条件構文と繰り返し構文を取り上げる。

June

15

17:15–20:30

文字列処理

バイオインフォマティクスの分野において、塩基配列やアミノ酸配列などの文字列からなるデータを扱うことが多い。第 2 回目の授業では、Python を利用した文字列処理を紹介し、FASTA や GFF などのファイルから情報を抽出する方法を取り上げる。

June

22

17:15–20:30

データ解析

Python で CSV や TSV ファイルに保存された数値データを扱うときに、NumPy や Pandas ライブラリーを使用する。第 3 回目の授業では、NumPy や Pandas の機能を紹介し、ベクトルや行列のデータ処理を中心に取り上げる。

June

29

17:15–20:30

データ可視化

Python で数値データを可視化するときに matplotlib ライブラリーを使用する。第 4 回目の授業では、matplotlib の基本的な使い方を紹介し、全回の授業を復習しながらデータの可視化を行う。

成績評価

出席

レポート課題

- 授業の始めに出席確認を 1 回だけを行う。出席 1 回につき 1 点を与える。
- 出席確認後、レポート課題を示す。毎回 3 問出題し、授業全体を通して合計 12 問出題する。1 問 8 点とする。
- 授業を聞かなくても、レポート課題を解けそうであれば、遠慮なく退室していただいて構いません。

レポート課題の提出方法

- レポート課題を Jupyter Notebook / Jupyter Lab 上で解き、そのファイル (.ipynb) をメールで提出する。

 **report@aabbdd.jp**

- 注意事項
 - メールの件名を「特論課題3」としてください。
 - メールの本文に名前を書いてください。
 - ファイルの名前を学生証番号（例：310000A.ipynb）としてください。
 - ファイル中に、名前・所属・研究内容などの個人情報・機密情報を書かないでください。個人情報・機密情報を含むレポートを零点とする。

レポート課題 (第 3 回)

問題 1

下図のように入力配列 a に対して、畳み込み演算と最大プーリング演算を行い、最後にその結果を 1 次元配列 x に変更せよ。なお、演算方法については問題 N2-1, N2-2 を参照のこと。

a

0	0	0	0	0	0	0	0
0	0	2	0	1	2	0	0
0	1	0	3	0	0	2	0
0	3	1	0	2	2	1	0
0	2	2	2	1	0	1	0
0	1	0	1	3	2	2	0
0	0	3	1	0	2	0	0
0	0	0	0	0	0	0	0

b

1	0	0
0	1	0
0	0	1

畳み込み演算



0	5	0	1	4	0
2	0	7	2	2	4
5	4	1	5	3	1
2	6	6	3	4	3
4	3	3	7	3	2
0	4	1	1	5	2

プーリング演算



5	7	4
6	6	4
4	7	5

次元変更



x

5
6
4
7
6
7
4
4
5

レポート課題 (第 3 回)

問題 2

rice.txt を読み込み、イネ wt およびイネ AUN842 を F10, NH4Cl, NH4NO3 の環境で栽培したときの shoot_dry_mass と root_dry_mass の平均値をすべて求めよ。なお、必要に応じて `groupby(['fert', 'variety'])` を使ってよい。

trt	fert	dry_mass	average
wt	F10	shoot_dry_mass	?
		root_dry_mass	?
	NH4Cl	shoot_dry_mass	?
		root_dry_mass	?
	NH4NO3	shoot_dry_mass	?
		root_dry_mass	?

trt	fert	dry_mass	average
AUN842	F10	shoot_dry_mass	?
		root_dry_mass	?
	NH4Cl	shoot_dry_mass	?
		root_dry_mass	?
	NH4NO3	shoot_dry_mass	?
		root_dry_mass	?

レポート課題 (第 3 回)

問題 3

表と裏の出る確率がそれぞれ 0.5 のコインを 1 枚用意する。コインを投げ、表が出たときに文字列 AA を書き、裏が出たときに文字 B を書く。例えば、コインを 5 回投げて、表、裏、裏、表、裏であったとすると、得られる文字列は AABBAAB となる。このとき、4 番目の文字は B、5 番目の文字は A である。

さて、コインを 10 回投げたとき、9 番目の文字が A で、かつ 10 番目の文字が B となる確率を、小数点 2 桁まで正確に求めよ。なお、必要であれば一様乱数を生成する関数を使っても良い。

農学生命情報科学特論 I

3

NumPy

Pandas

画像解析



アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

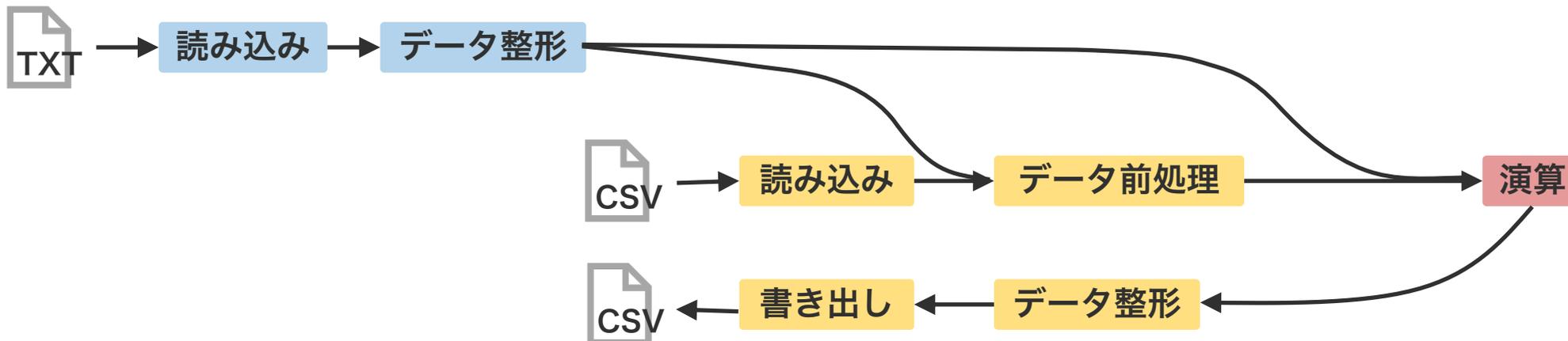
- リスト
- 2次元リスト

CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2次元配列



NumPy

- 1 次元配列
- 2 次元配列
- 3 次元配列
- データの読み書き

1 次元配列 / np.array

NumPy の 1 次元配列は、Python の 1 次元リストを拡張したようなオブジェクトである。NumPy 配列は、NumPy で用意された関数を使用して作成する。例えば、np.array 関数を使えば、任意の Python のリストを配列に変換できる。また、np.full や np.linspace などの関数を使えば、すべての要素が同じ値であるような配列や等差数列のような配列を作することもできる。

np.array

np.zeros

np.ones

np.full

np.arange

np.linspace

```
import numpy as np
```

```
a = [1, 1, 2, 3, 5, 8]
```

```
a
```

```
# [1, 1, 2, 3, 5, 8]
```

```
b = np.array(a)
```

```
b
```

```
# array([1, 1, 2, 3, 5, 8])
```

```
c = np.array([1, 1, 2, 3, 5, 8])
```

```
c
```

```
# array([1, 1, 2, 3, 5, 8])
```

1 次元配列 / np.full

すべての要素が同じ値であるような配列を作るとき、`np.zeros`, `np.ones`, `np.full` 関数を使用すると便利である。`np.zeros` と `np.ones` 関数は、関数名の通り、すべての要素が 0 または 1 であるような配列を作るときに使用する。また、また、`np.full` 関数は、任意の値で初期化された配列を作るときに使用する。

`np.array`

`np.zeros`

`np.ones`

`np.full`

`np.arange`

`np.linspace`

```
import numpy as np
```

```
a = np.zeros(5)
```

```
a  
# array([0, 0, 0, 0, 0])
```

```
b = np.ones(8)
```

```
b  
# array([1, 1, 1, 1, 1, 1, 1, 1])
```

```
c = np.full(5, np.nan)
```

```
c  
# array([nan, nan, nan, nan, nan])
```

1 次元配列 / np.arange

等差数列からなる配列を作るとき、np.arange 関数を使用する。np.arange 関数は、start (数列の最初の値), stop (数列の範囲), step (間隔) の 3 つの引数を受け取り、それらに基づいて等差数列からなる配列を作る。start を省略すると start=0 となり、step を省略すると step=1 となる。

np.array

np.zeros

np.ones

np.full

np.arange

np.linspace



関数名に注意。np.arrange ではない。NumPy 前身であった Numeric 時代で使われた arange 関数の省略形として arange が使われていたことに由来する。

```
import numpy as np

a = np.arange(5)
a
# array([0, 1, 2, 3, 4])

b = np.arange(1, 6)
b
# array([1, 2, 3, 4, 5])

c = np.arange(1, 6, 2)
c
# array([1, 3, 5])

d = np.arange(10, 0, -2)
d
# array([10, 8, 6, 4, 2])
```

1 次元配列 / np.linspace

等差数列からなる配列を作るとき、`np.linspace` 関数も利用できる。この関数は、`start` (数列の最初の値), `stop` (数列の最後の値), `num` (要素数) の 3 つの引数を受け取り、それらに基づいて等差数列からなる配列を作る。`num` を省略すると `num=50` となる。

`np.array`

`np.zeros`

`np.ones`

`np.full`

`np.arange`

`np.linspace`



関数名に注意。`np.linspace` ではない。`Linearly space vectors` を生成する関数のため、`linspace` である。

```
import numpy as np
```

```
a = np.linspace(1, 9, 3)
a
# array([1., 5., 9.])
```

```
b = np.linspace(1, 9, 5)
b
# array([1., 3., 5., 7., 9.])
```

```
c = np.linspace(1, 0, 5)
c
# array([1.   , 0.75, 0.5  , 0.25, 0.  ])
```

1 次元配列 / 数値計算

NumPy には、切り上げや切り捨てを行う `np.ceil` や `np.floor`、対数化を行う `np.log`、三角関数の値を計算する `np.sin`, `np.cos` や `np.tan` など、平均や分散を計算する `np.mean` や `np.std` など、多くの関数が用意されている。これらの関数名をすべて覚える必要はなく、使いたいときにその使い方を調べればよい。

```
import numpy as np

a = np.array([1, 5, 10, 50, 100])

np.log10(a)
# array([0., 0.69897, 1., 1.69897, 2.])

np.sqrt(a)
# array([1., 2.2360, 3.1622, 7.0710, 10.])

np.mean(a)
# 33.2

np.std(a)
# 37.722142038860945
```

1 次元配列

NumPy の 1 次元配列は、リストほぼ同じように取り扱うことができる。配列は基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることもできる。その際、配列の先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。

```
import numpy as np

w = np.array([12, 10, 11, 13, 11])

w[0]
# 12

w[1]
# 10

w[2]
# 11

w[2] = 9

w
# array([12, 10, 9, 13, 11])
```



1 次元配列 / スライス

Python のリストと同様に、隣り合う要素をスライスして取得することもできる。スライスを行うとき、スライスの開始位置と終了（手前の）位置をコロン（:）で区切って指定する。

スライスにより取り出された部分配列は、元の配列への参照となっている。したがって、スライスで得られた部分配列の値を変更すると、元の配列の値も変化してしまう。

```
a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])
```

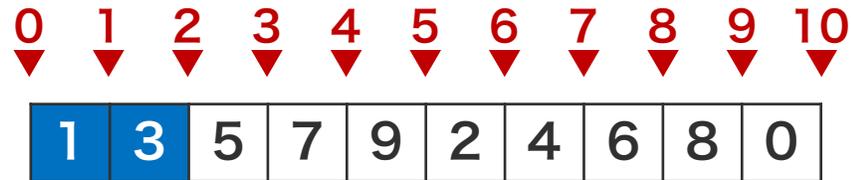
```
b = a[3:6]
b
# np.array([7, 9, 2])
```

```
b[0] = 0
b[1] = 0
b[2] = 0
```

```
b
# np.array([0, 0, 0])
```

```
a
# np.array([1, 3, 5, 0, 0,
#           0, 4, 6, 8, 0])
```

1 次元配列 / スライス



```
a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])
```

```
a[0:2]
# np.array([1, 3])
```

```
a[2:6]
# np.array([5, 7, 9, 2])
```

```
a[:4]
# np.array([1, 3, 5, 7])
```

```
a[5:]
# np.array([2, 4, 6, 8, 0])
```

1 次元配列 / スライス

スライスを行うとき、開始位置と終了位置の他に、ステップ数を与えることもできる。1 要素おきに値を 1 つ取り出す場合などに便利である。また、配列では、連続していない要素を同時に取り出すこともできる。

```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])

a[2:9]

a[2:9:1]

a[2:9:3]

a[[0, 2, 4, 6]]
```

1 次元配列 / スライス

0 1 2 3 4 5 6 7 8 9 10
▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼

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

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

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

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

```
import numpy as np
```

```
a = np.array([1, 3, 5, 7, 9,  
             2, 4, 6, 8, 0])
```

```
a[2:9]  
# array([5, 7, 9, 2, 4, 6, 8])
```

```
a[2:9:1]  
# array([5, 7, 9, 2, 4, 6, 8])
```

```
a[2:9:3]  
# array([5, 2, 8])
```

```
a[[0, 2, 4, 6]]  
# array([1, 5, 9, 4])
```

1 次元配列 / フィルター

配列から要素を取り出すとき、位置番号で指定するほか、True または False からなる配列（ブーリアンベクトル）で指定することもできる。このとき、ブーリアンベクトルの長さは、操作対象となる配列の長さと同じでなければならない。

```
import numpy as np

a = np.array([ 2, 4, 6, 8])
k = np.array([True, True, False, True])
```

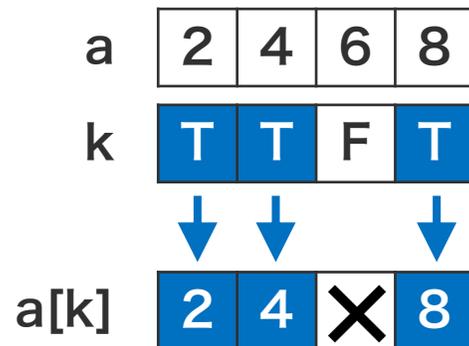
```
a[k]
```

```
a = np.array([ 2, 4, 6, 8])
k = np.array([False, True, True, True])
```

```
a[k]
```

1 次元配列 / フィルター

フィルター k を用いて、ベクトル a の要素をフィルタリングしているイメージ。



```
import numpy as np
```

```
a = np.array([ 2, 4, 6, 8])
```

```
k = np.array([True, True, False, True])
```

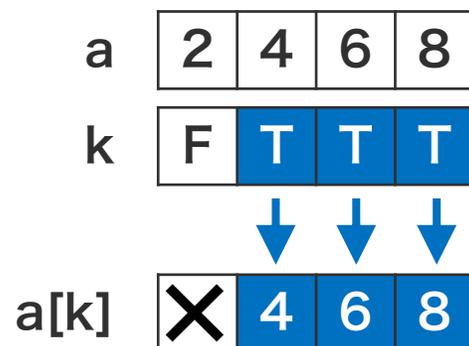
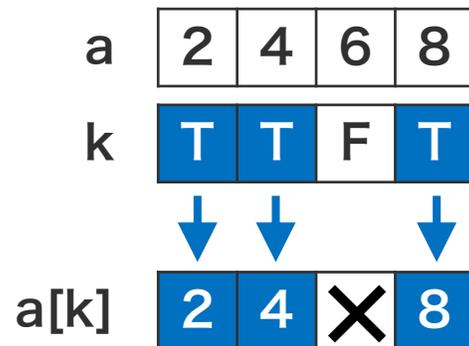
```
a[k]  
# array([2, 4, 8])
```

```
a = np.array([ 2, 4, 6, 8])
```

```
k = np.array([False, True, True, True])
```

```
a[k]
```

1 次元配列 / フィルター



フィルター k を用いて、ベクトル a の要素をフィルタリングしているイメージ。

```
import numpy as np
```

```
a = np.array([ 2, 4, 6, 8])
```

```
k = np.array([True, True, False, True])
```

```
a[k]  
# array([2, 4, 8])
```

```
a = np.array([ 2, 4, 6, 8])
```

```
k = np.array([False, True, True, True])
```

```
a[k]  
# array([4, 6, 8])
```

1 次元配列 / フィルター

フィルターは、True または False を組み合わせて直打ちで作成できるが、ある条件を制定して、その条件に基づいて作成することが一般的である。例えば、5 よりも大きい値を取り出すフィルターや奇数の値を取り出すフィルターは、右のように作成する。

```
import numpy as np

a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

f1 = (a > 5)
f1
# array([F, F, T, T, F, F, F, T])

f2 = (a % 2 == 1)
f2
# array([F, F, F, F, T, T, T, T])
```

1 次元配列 / フィルター

0	1	2	3	4	5	6	7	8
▼	▼	▼	▼	▼	▼	▼	▼	▼
2	4	6	8	1	3	5	7	

f1

F	F	T	T	F	F	F	T
2	4	6	8	1	3	5	7

f2

F	F	F	F	T	T	T	T
2	4	6	8	1	3	5	7

```
import numpy as np
```

```
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])
```

```
f1 = (a > 5)
```

```
a[f1]
```

```
# array([6, 8, 7])
```

```
f2 = (a % 2 == 1)
```

```
a[f2]
```

```
# array([1, 3, 5, 7])
```

1 次元配列 / フィルター

0	1	2	3	4	5	6	7	8
▼	▼	▼	▼	▼	▼	▼	▼	▼
2	4	6	8	1	3	5	7	

f1

F	F	T	T	F	F	F	T
2	4	6	8	1	3	5	7

f2

F	F	F	F	T	T	T	T
2	4	6	8	1	3	5	7

a < 4

T	F	F	F	T	T	F	F
2	4	6	8	1	3	5	7

```
import numpy as np
```

```
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])
```

```
f1 = (a > 5)
```

```
a[f1]
```

```
# array([6, 8, 7])
```

```
f2 = (a % 2 == 1)
```

```
a[f2]
```

```
# array([1, 3, 5, 7])
```

```
a[(a < 4)]
```

```
# array([2, 1, 3])
```

フィルターを一次変数に保存せずに、直接使用することもできる。

1 次元配列 / フィルター

複数のフィルターを重ねて使用することもできる。この場合、フィルターを重ねるときに AND 演算で重ねるか、OR 演算で重ねるかを指定する必要がある。

```
import numpy as np  
  
a = np.array([1, 3, 5, 7, 9,  
             2, 4, 6, 8, 0])
```

```
f1 = (a % 2 == 1)
```

```
f2 = (a > 5)
```

```
a[f1 & f2]
```

```
a[f1 | f2]
```

1 次元配列 / フィルター

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

f1

--	--	--	--	--	--	--	--	--	--

f2

--	--	--	--	--	--	--	--	--	--

f1 & f2

--	--	--	--	--	--	--	--	--	--

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

f1 | f2

--	--	--	--	--	--	--	--	--	--

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

```
import numpy as np
```

```
a = np.array([1, 3, 5, 7, 9,  
             2, 4, 6, 8, 0])
```

```
f1 = (a % 2 == 1)
```

```
f2 = (a > 5)
```

```
a[f1 & f2]
```

```
a[f1 | f2]
```

1 次元配列 / フィルター

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

f1

T	T	T	T	T	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---

f2

F	F	F	T	T	F	F	T	T	F
---	---	---	---	---	---	---	---	---	---

f1 & f2

--	--	--	--	--	--	--	--	--	--

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

f1 | f2

--	--	--	--	--	--	--	--	--	--

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

```
import numpy as np
```

```
a = np.array([1, 3, 5, 7, 9,  
              2, 4, 6, 8, 0])
```

```
f1 = (a % 2 == 1)
```

```
f2 = (a > 5)
```

```
a[f1 & f2]
```

```
a[f1 | f2]
```

1 次元配列 / フィルター

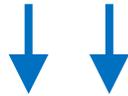
1	3	5	7	9	2	4	6	8	0
---	---	---	---	---	---	---	---	---	---

f1

T	T	T	T	T	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---

f2

F	F	F	T	T	F	F	T	T	F
---	---	---	---	---	---	---	---	---	---



f1 & f2

F	F	F	T	T	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---

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

```
import numpy as np
```

```
a = np.array([1, 3, 5, 7, 9,  
             2, 4, 6, 8, 0])
```

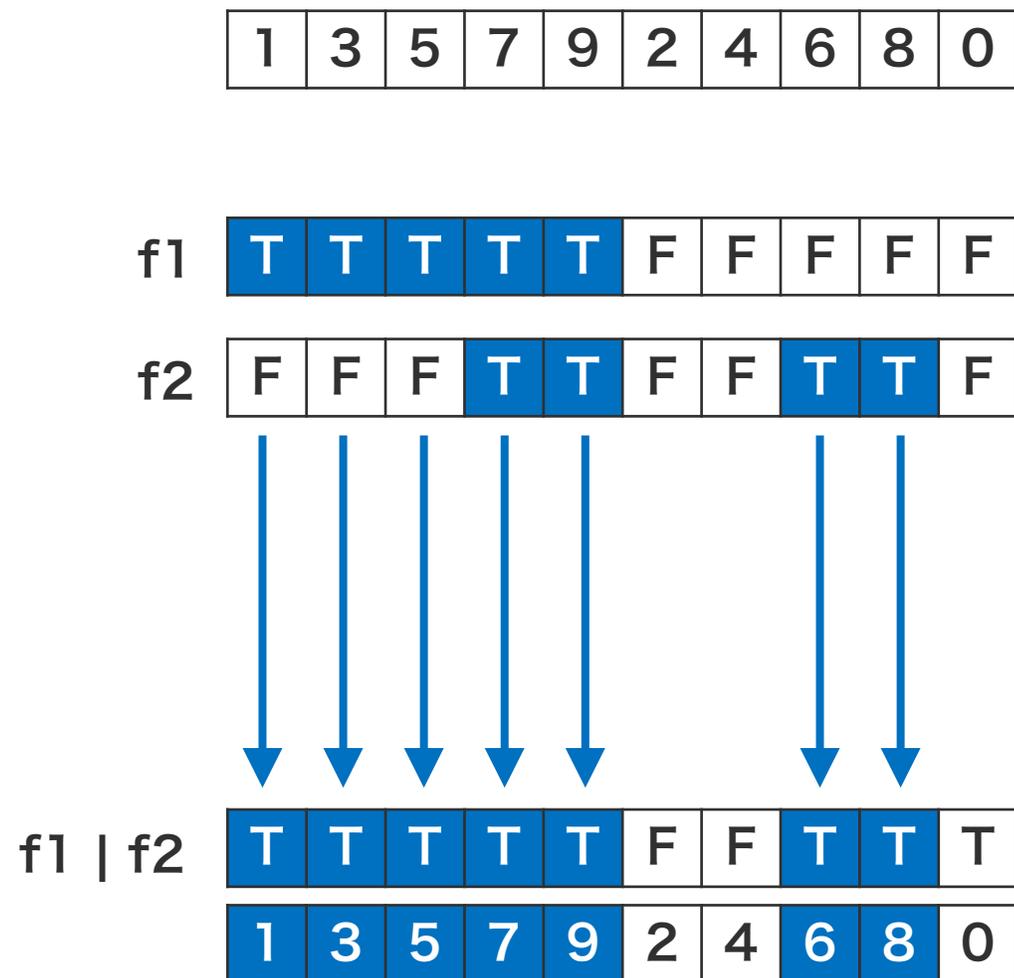
```
f1 = (a % 2 == 1)
```

```
f2 = (a > 5)
```

```
a[f1 & f2]  
# array([7, 9])
```

```
a[f1 | f2]
```

1 次元配列 / フィルター



```
import numpy as np
```

```
a = np.array([1, 3, 5, 7, 9,  
             2, 4, 6, 8, 0])
```

```
f1 = (a % 2 == 1)
```

```
f2 = (a > 5)
```

```
a[f1 & f2]  
# array([7, 9])
```

```
a[f1 | f2]  
# array([1, 3, 5, 7, 9, 6, 8])
```

集合演算

NumPy の 1 次元配列に対して、集合演算を行う関数が用意されている。与えられた 2 つの 1 次元配列を比較し、共通要素を取り出したり、一方に含まれかつ他方に含まれていないような要素を取り出したりすることができる。

メソッド	動作
<code>.union1d(a,b)</code>	集合 a と集合 b の和集合
<code>.setdiff1d(a,b)</code>	集合 a と集合 b の差集合
<code>.intersect1d(a,b)</code>	集合 a と集合 b の積集合
<code>.setxor1d(a,b)</code>	集合 a と集合 b の対象差集合
<code>.in1d(a,b)</code>	集合 a の各要素が集合 b に含まれているかどうかを判定

```
import numpy as np

a = np.array([1, 1, 2, 3, 5, 8])
b = np.array([2, 4, 6, 8])

np.union1d(a, b)
# array([1, 2, 3, 4, 5, 6, 8])

np.setdiff1d(a, b)
# array([1, 3, 5])

np.setdiff1d(b, a)
# array([4, 6])
```

擬似乱数生成

NumPy の random モジュールには擬似乱数を生成する機能が実装されている。次の表に、一様分布および正規分布から乱数を生成する関数を示した。これ以外にも、ポアソン分布やガンマ分布などの分布から乱数を生成する関数が多数用意されている。必要なときに調べて使うとよい。

メソッド	動作
<code>.rand(n)</code>	範囲 $[0, 1)$ の一様分布から n 個の乱数を生成。
<code>.normal(m, s, n)</code>	平均 m , 標準偏差 s の正規分布から n 個の乱数を生成。
<code>.randint(l, h, n)</code>	範囲 $[l, h)$ から n 個の整数乱数を生成。
<code>.shuffle(arr)</code>	配列 <code>arr</code> の要素をシャッフルする。
<code>.seed(s)</code>	乱数シードを s に固定する。

```
import numpy as np

np.random.seed(2020)

np.random.rand(3)
# array([0.986276, 0.873391, 0.509745])

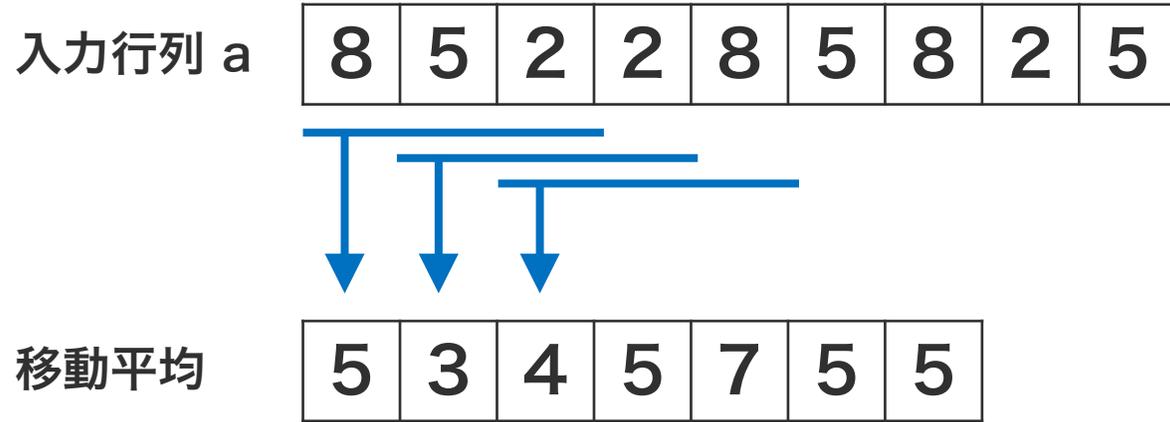
np.random.normal(5, 1, 3)
# array([4.115677, 3.762758, 4.291560])

np.random.randint(0, 2, 5)
# array([0, 1, 1, 1, 1])

a = np.array([1, 2, 3, 4, 5, 6, 7])
np.random.shuffle(a)
a
# array([2, 7, 6, 5, 1, 4, 3])
```

問題 N1-1

入力配列 a に対して 3 時点の移動平均を求めよ。

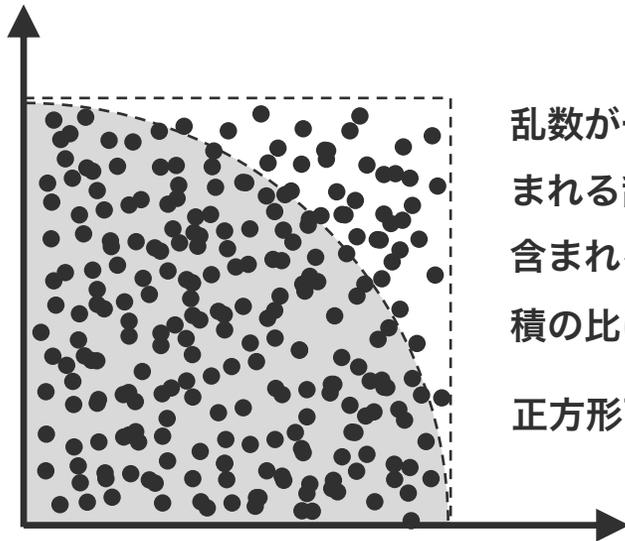


```
import numpy as np

a = np.array([8, 5, 2, 2, 8,
              5, 8, 2, 5])
```

問題 N1-2

`np.random.rand(n)` 関数は、 $0 \leq x < 1$ の一様分布から n 個の乱数を生成する関数である。 $0 \leq x < 1$ および $0 \leq y < 1$ の範囲で乱数を生成し、下図のように、正方形に含まれる乱数の個数と、第一象限にある単位円の内側に含まれる乱数の個数に着目して、円周率を小数 3 桁 (= 3.141...) まで正確に求めよ。



乱数が一様であれば、正方形内部に含まれる乱数の個数と単位円灰色部分に含まれる乱数の個数の比が、両者の面積の比に近似できる。

$$\text{正方形面積} : \text{単位円灰色部分} = 1 : \frac{\pi}{4}$$

```
import numpy as np
```

```
n = 10000
```

```
x = np.random.rand(n)
```

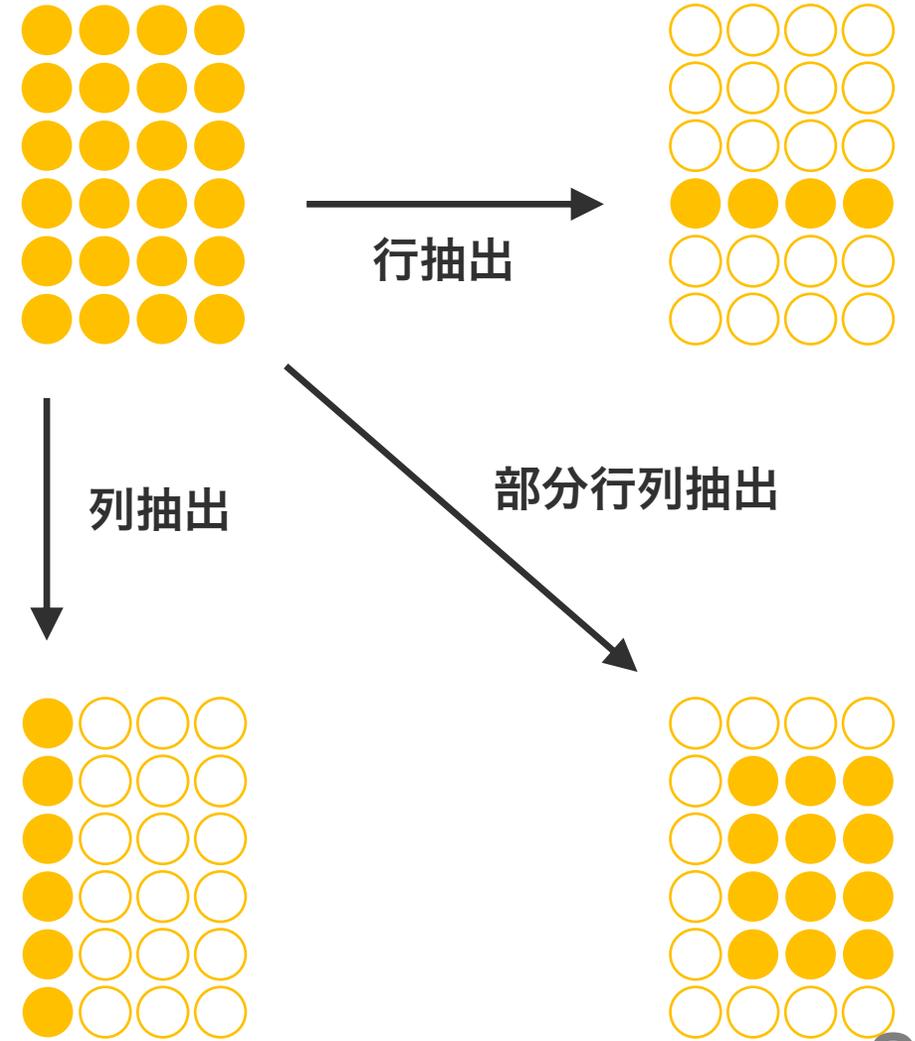
```
y = np.random.rand(n)
```

NumPy

- 1 次元配列
- 2 次元配列
- 3 次元配列
- データの読み書き

2次元配列

NumPy の2次元配列は、縦と横の構造を持ち、数学の行列のように、特定の行あるいは列を抽出したり、行列演算を行ったりすることができる。



2次元配列 / np.array

2次元配列を作成するには、2次元のリストを作成して、それを np.array 関数に代入することで作成する。

```
a = np.array([[11, 12, 13, 14, 15, 16],  
              [21, 22, 23, 24, 25, 26],  
              [31, 32, 33, 34, 35, 36],  
              [41, 42, 43, 44, 45, 46],  
              [51, 52, 53, 54, 55, 56],  
              [61, 62, 63, 64, 65, 66]])
```

11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46
51	52	53	54	55	56
61	62	63	64	65	66

2次元配列 / np.zeros, np.ones

2次元配列も1次元配列と同様に、同じ値からなる配列を作る場合、np.zeros、np.ones、np.fullなどの関数を使う。なお、2次元配列の場合、横と縦のサイズを指定する必要がある。

```
b = np.zeros((2, 5))
```

0	0	0	0	0
0	0	0	0	0

```
b = np.ones((5, 3))
```

1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

2次元配列 / np.full

2次元配列も1次元配列と同様に、同じ値からなる配列を作る場合、`np.zeros`、`np.ones`、`np.full`などの関数を使う。なお、2次元配列の場合、横と縦のサイズを指定する必要がある。

```
a = np.full((5, 3), np.nan)
```

nan	nan	nan

2次元配列 / np.identity

2次元配列の場合、np.identity 関数を使用して、単位行列を作成することができる。

```
a = np.identity(6)
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

2次元配列

2次元行列の構造を調べるときは、`ndim`, `shape`, `np.size` などを利用する。`ndim` 属性には、配列の次元数が記録されている。`shape` 属性には、配列構造（サイズ）が記録されている。また、`np.size` 関数を使用することで、配列の特定の次元のサイズを取得することができる。

11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46

```
a = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46]])
```

```
a.ndim
# 2
```

```
a.shape
# (4, 6)
```

```
np.size(a, axis=0)
# 4
```

```
np.size(a, axis=1)
# 6
```

2次元配列

2次元配列から要素を取り出すとき、位置番号を指定して取り出したり、スライスして取り出したりすることができる。

```
b = np.array([[11, 12, 13, 14, 15, 16],  
              [21, 22, 23, 24, 25, 26],  
              [31, 32, 33, 34, 35, 36],  
              [41, 42, 43, 44, 45, 46],  
              [51, 52, 53, 54, 55, 56],  
              [61, 62, 63, 64, 65, 66]])
```

```
b[1, 2]
```

```
b[1, 2:5]
```

```
b[1:5, 3]
```

```
b[2:4, 1:6]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

```
b[1, 2]
```

```
# 23
```

```
b[1, 2:5]
```

```
b[1:5, 3]
```

```
b[2:4, 1:6]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
b[1, 2]  
# 23
```

```
b[1, 2:5]  
# array([23, 24, 25])
```

```
b[1:5, 3]
```

```
b[2:4, 1:6]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
b[1, 2]  
# 23
```

```
b[1, 2:5]  
# array([23, 24, 25])
```

```
b[1:5, 3]  
# array([24, 34, 44, 54])
```

```
b[2:4, 1:6]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

```
b[1, 2]
# 23
```

```
b[1, 2:5]
# array([23, 24, 25])
```

```
b[1:5, 3]
# array([24, 34, 44, 54])
```

```
b[2:4, 1:6]
# array([[32, 33, 34, 35, 36],
#        [42, 43, 44, 45, 46]])
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
b[3:5, 2:]
```

```
b[3:5, :]
```

```
b[:, 2]
```

```
b[:, 1:3]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

```
b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
```

```
b[:, 2]
```

```
b[:, 1:3]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
b[3:5, 2:]  
# array([[43, 44, 45, 46],  
#        [53, 54, 55, 56]])  
b[3:5, :]  
# array([[41, 42, 43, 44, 45, 46],  
#        [51, 52, 53, 54, 55, 56]])  
b[:, 2]
```

```
b[:, 1:3]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
b[3:5, 2:]  
# array([[43, 44, 45, 46],  
#        [53, 54, 55, 56]])  
b[3:5, :]  
# array([[41, 42, 43, 44, 45, 46],  
#        [51, 52, 53, 54, 55, 56]])  
b[:, 2]  
# array([13, 23, 33, 43, 53, 63])
```

```
b[:, 1:3]
```

2次元配列

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

```
b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
```

```
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
```

```
b[:, 2]
# array([13, 23, 33, 43, 53, 63])
```

```
b[:, 1:3]
# array([[12, 13], [22, 23], [32, 33],
#        [42, 43], [52, 53], [62, 63]])
```

2 次元配列 / 注意点

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
r = [0, 2, 4]
```

```
c = [1, 3, 5]
```

```
b[r, c]
```

2 次元配列 / 注意点

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
r = [0, 2, 4]
```

```
c = [1, 3, 5]
```

```
b[r, c]
```

2 次元配列 / 注意点

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
r = [0, 2, 4]
```

```
c = [1, 3, 5]
```

```
b[r, c]
```

```
# array([12, 34, 56])
```

2 次元配列 / 注意点

	0	1	2	3	4	5	6
0	11	12	13	14	15	16	
1	21	22	23	24	25	26	
2	31	32	33	34	35	36	
3	41	42	43	44	45	46	
4	51	52	53	54	55	56	
5	61	62	63	64	65	66	
6							

```
b = np.array([[11, 12, 13, 14, 15, 16],  
             [21, 22, 23, 24, 25, 26],  
             [31, 32, 33, 34, 35, 36],  
             [41, 42, 43, 44, 45, 46],  
             [51, 52, 53, 54, 55, 56],  
             [61, 62, 63, 64, 65, 66]])
```

```
r = [0, 2, 4]  
c = [1, 3, 5]
```

```
b[r, c]  
# array([12, 34, 56])
```

```
b[np.ix_(r, c)]  
# array([[12, 14, 16],  
#        [32, 34, 36],  
#        [52, 54, 56]])
```

行列計算

NumPy の 2 次元配列に、行列演算が定義されている。配列の各要素同士の加減乗除の他に、内積や外積も簡単に求めることができる。

計算式	計算内容
$a + b$	各要素の足し算
$a - b$	各要素の引き算
$a * b$	各要素の掛け算 (アダマール積)
a / b	各要素の割り算
<code>np.dot(a, b)</code>	行列同士の内積
<code>np.outer(a, b)</code>	行列同士の外積 (テンソル積)
<code>np.sum(a)</code>	全要素の和
<code>a.T</code>	行列 a の転置行列

```
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

b = np.array([[1, 1, 1],
              [0, 1, 1],
              [0, 0, 1]])

a + b
# array([[ 2,  3,  4],
#        [ 4,  6,  7],
#        [ 7,  8, 10]])

np.dot(a, b)
# array([[ 1,  3,  6],
#        [ 4,  9, 15],
#        [ 7, 15, 24]])
```

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])
```

```
b = np.array([[1, 0, 0],
              [0, 1, 0],
              [0, 0, 1]])
```

入力行列 a

カーネル行列 b

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

1	0	0
0	1	0
0	0	1

$$\begin{aligned} &0*1 + 2*0 + 0*0 + \\ &1*0 + 0*1 + 3*0 + \\ &3*0 + 1*0 + 0*1 = 0 \end{aligned}$$

演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],  
              [1, 0, 3, 0, 0, 2],  
              [3, 1, 0, 2, 2, 1],  
              [2, 2, 2, 1, 0, 1],  
              [1, 0, 1, 3, 2, 2],  
              [0, 3, 1, 0, 2, 0]])
```

```
b = np.array([[1, 0, 0],  
              [0, 1, 0],  
              [0, 0, 1]])
```

入力行列 a

カーネル行列 b

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

1	0	0
0	1	0
0	0	1

$$\begin{aligned} & 2*1 + 0*0 + 1*0 + \\ & 0*0 + 3*1 + 0*0 + \\ & 1*0 + 0*0 + 2*1 = 7 \end{aligned}$$

演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],  
              [1, 0, 3, 0, 0, 2],  
              [3, 1, 0, 2, 2, 1],  
              [2, 2, 2, 1, 0, 1],  
              [1, 0, 1, 3, 2, 2],  
              [0, 3, 1, 0, 2, 0]])
```

```
b = np.array([[1, 0, 0],  
              [0, 1, 0],  
              [0, 0, 1]])
```

入力行列 a

カーネル行列 b

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

1	0	0
0	1	0
0	0	1

$$\begin{aligned} &0*1 + 1*0 + 2*0 + \\ &3*0 + 0*1 + 0*0 + \\ &0*0 + 2*0 + 2*1 = 2 \end{aligned}$$

演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-2

行列 a に対して 2x2 範囲の最大プーリング演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],  
              [1, 0, 3, 0, 0, 2],  
              [3, 1, 0, 2, 2, 1],  
              [2, 2, 2, 1, 0, 1],  
              [1, 0, 1, 3, 2, 2],  
              [0, 3, 1, 0, 2, 0]])
```

入力行列 a

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

演算結果

2	3	2
3	2	2
3	3	2

問題 N2-2

行列 a に対して 2x2 範囲の最大プーリング演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],  
              [1, 0, 3, 0, 0, 2],  
              [3, 1, 0, 2, 2, 1],  
              [2, 2, 2, 1, 0, 1],  
              [1, 0, 1, 3, 2, 2],  
              [0, 3, 1, 0, 2, 0]])
```

入力行列 a

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

演算結果

2	3	2
3	2	2
3	3	2

問題 N2-2

行列 a に対して 2x2 範囲の最大プーリング演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])
```

入力行列 a

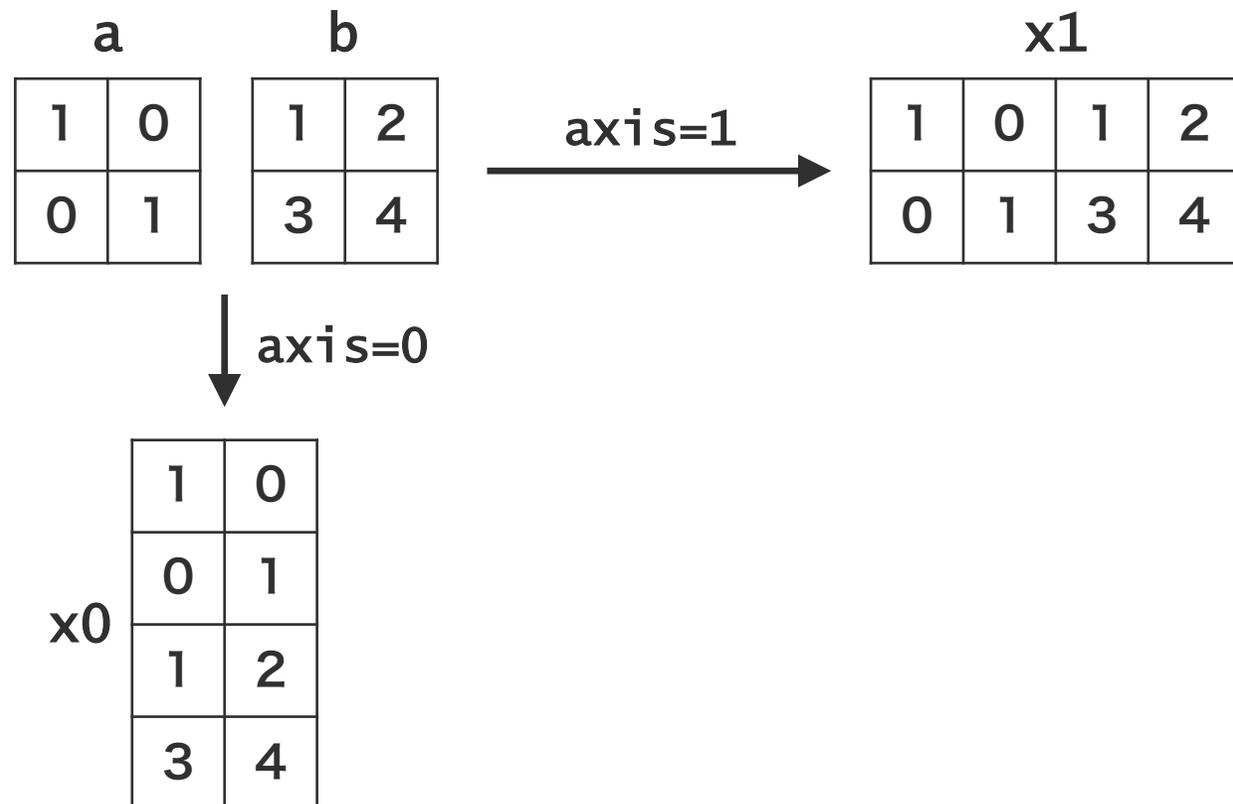
0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

演算結果

2	3	2
3	2	2
3	3	2

行列操作 / np.concatenate

np.concatenate 関数は、複数のNumPy 配列を結合する機能を持つ。この関数を使うとき、結合する軸（次元）方向を axis 引数で指定。省略された場合は axis=0 となる。



```
a = [[1, 0], [0, 1]]
b = [[1, 2], [3, 4]]
```

```
x0 = np.concatenate([a, b], axis=0)
x0
# array([[1, 0],
#        [0, 1],
#        [1, 2],
#        [3, 4]])
```

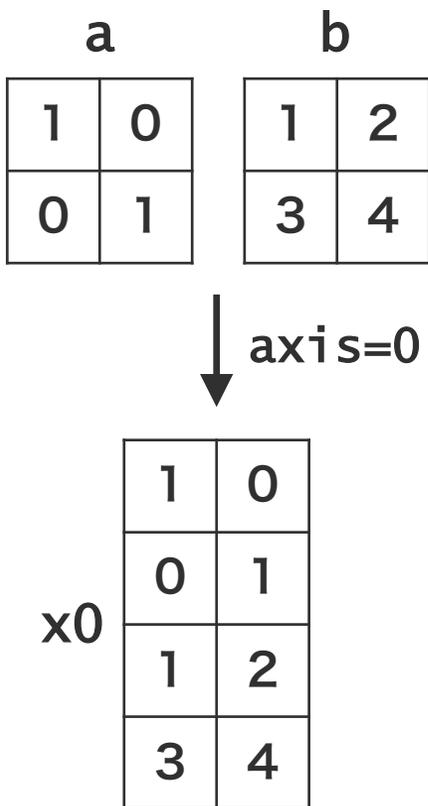
```
x1 = np.concatenate([a, b], axis=1)
x1
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])
```

```
c = [9, 9]
```

```
y0 = np.concatenate([a, c], axis=0)
# ValueError: all the input arrays must
# have same number of dimensions, ...
```

行列操作 / np.vstack

np.vstack 関数は複数の配列を第 1 次元方向に結合する機能を持つ。axis=0 のときの np.concatenate 関数の機能と同じ。



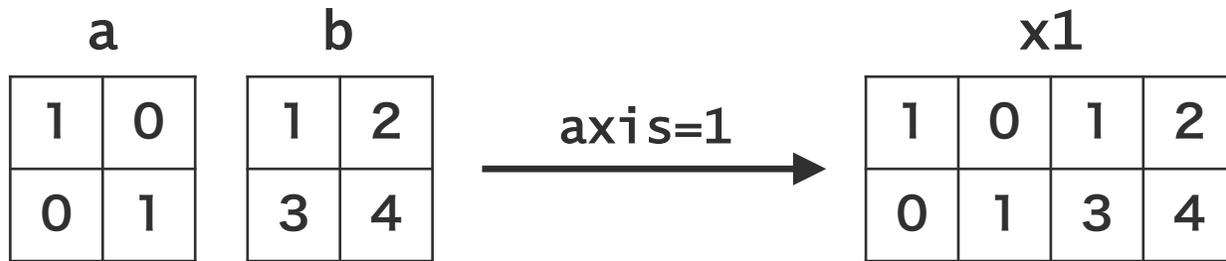
```
a = [[1, 0], [0, 1]]  
b = [[1, 2], [3, 4]]
```

```
x = np.concatenate([a, b], axis=0)  
x  
# array([[1, 0],  
#        [0, 1],  
#        [1, 2],  
#        [3, 4]])
```

```
y = np.vstack([a, b])  
y  
# array([[1, 0],  
#        [0, 1],  
#        [1, 2],  
#        [3, 4]])
```

行列操作 / np.hstack

np.hstack 関数は複数の配列を第 2 次元方向に結合する機能を持つ。axis=1 のときの np.concatenate 関数の機能と同じ。



```
a = [[1, 0], [0, 1]]
b = [[1, 2], [3, 4]]
```

```
x = np.concatenate([a, b], axis=1)
x
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])
```

```
y = np.hstack([a, b])
y
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])
```

行列操作 / flatten

flatten 関数は、多次元配列を 1 次元配列に変更する関数である。多次元配列を崩すときに、横方向から崩すのか (order='C')、縦方向から崩すのか (order='F') を指定することもできる。

```
a = np.array([[1, 0, 2, 4],
              [0, 1, 3, 9]])

x = a.flatten()
x
# array([1, 0, 2, 4, 0, 1, 3, 9])

y = a.flatten(order='F')
y
# array([1, 0, 0, 1, 2, 3, 4, 9])
```

行列操作 / reshape

reshape 関数は、多次元配列の構造を変形する関数である。reshape の 1 つ目の引数に変形後の構造をリストで指定する。また、order 引数を指定することで、多次元配列のデータを読み取る方向を指定することもできる。

reshape 関数を利用して変形した配列は、元の配列のビューまたはコピーとなっている。メモリー容量や配列のサイズによってビューかコピーが決定される。したがって、変形後の配列の値を変更すると、変形元の配列の値も変わる可能性がある。

```
a = np.array([[1, 2, 2, 3],
              [0, 1, 2, 6],
              [0, 0, 1, 9]])

y = a.reshape([12])
y
## array([1, 2, 2, 3, 0, 1, 2,
##        6, 0, 0, 1, 9])

x = a.reshape([4, 3])
x
## array([[1, 2, 2],
##        [3, 0, 1],
##        [2, 6, 0],
##        [0, 1, 9]])
```

行列操作 / reshape

reshape 関数は、多次元配列の構造を変形する関数である。reshape の 1 つ目の引数に変形後の構造をリストで指定する。また、order 引数を指定することで、多次元配列のデータを読み取る方向を指定することもできる。

reshape 関数を利用して変形した配列は、元の配列のビューまたはコピーとなっている。メモリー容量や配列のサイズによってビューかコピーが決定される。したがって、変形後の配列の値を変更すると、変形元の配列の値も変わる可能性がある。

```
a = np.array([[1, 2, 2, 3],
              [0, 1, 2, 6],
              [0, 0, 1, 9]])
```

```
y = a.reshape([12], order='F')
y
## array([1, 0, 0, 2, 1, 0, 2,
##        2, 1, 3, 6, 9])
```

```
x = a.reshape([4, 3], order='F')
x
## array([[1, 1, 1],
##        [0, 0, 3],
##        [0, 2, 6],
##        [2, 2, 9]])
```

行列操作 / reshape

reshape 関数にて、変形後の配列の構造を指定するときに、-1 を指定することができる。-1 で指定された次元のサイズは、他の次元のサイズから自動的に計算される。

```
a = np.array([[1, 2, 2, 3],
              [0, 1, 2, 6],
              [0, 0, 1, 9]])
```

```
y1 = a.reshape([2, 2, 3])
```

```
y1
# array([[[1, 2, 2],
#         [3, 0, 1]],
#        [[2, 6, 0],
#         [0, 1, 9]]])
```

```
y2 = a.reshape([2, 2, -1])
```

```
y2
# array([[[1, 2, 2],
#         [3, 0, 1]],
#        [[2, 6, 0],
#         [0, 1, 9]]])
```

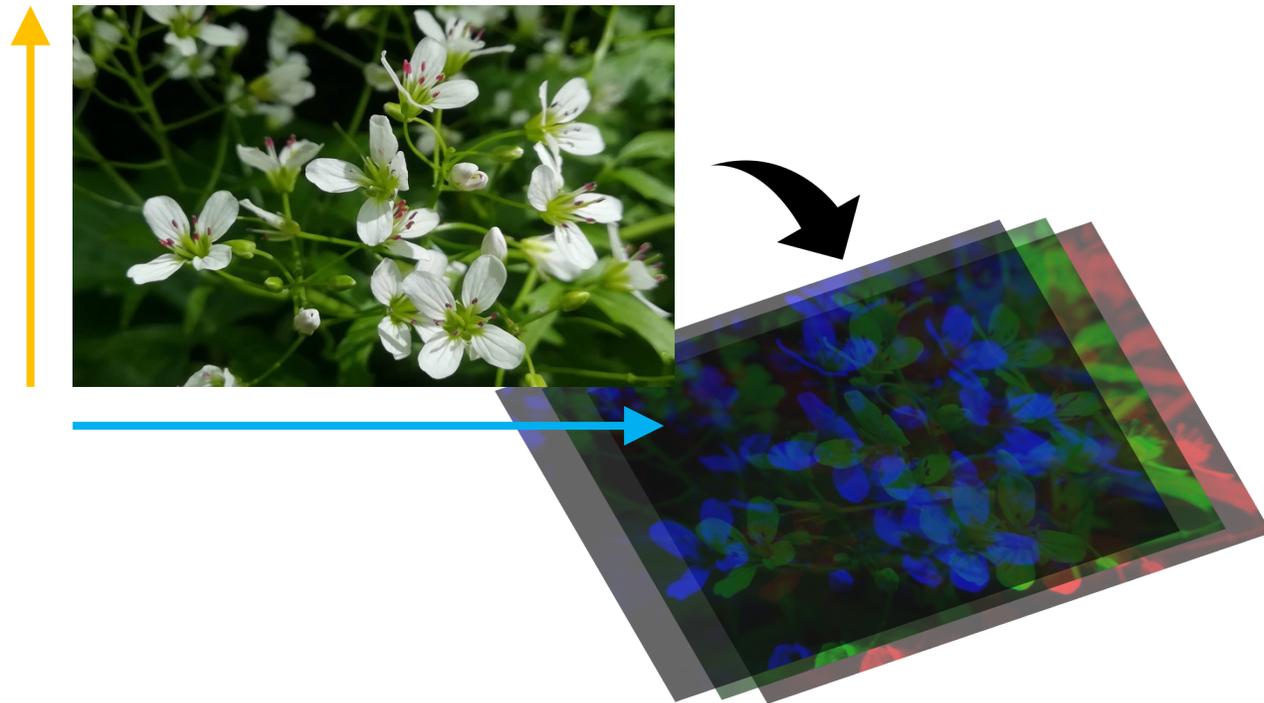
a は 12 要素。最初の 2 次元に 2 要素ずつを配置した場合、最後の次元は自動的に 3 になる。

NumPy

- 1 次元配列
- 2 次元配列
- 3 次元配列
- データの読み書き

3次元配列

3次元配列は、画像解析のときによく使われる。デジタルカメラで撮られている画像の色は、赤・緑・青の3原色によって表される。そのため、縦 n 横 m の画像は、 $n \times m$ の配列がまずあり、その各 (n, m) 要素にはRGBの3つの要素が含まれているようになる。



```
b = np.array([[1, 0, 1], [1, 1, 0], [1, 1, 1]],
              [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
              [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
              [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
              [[1, 0, 1], [1, 1, 0], [1, 1, 1]])
b
array([[1 0 1]
       [1 1 0]
       [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]])
```

NumPy

- 1 次元配列
- 2 次元配列
- 3 次元配列
- データの読み書き

データの読み込み / バイナリー

NumPy のバイナリデータを保存しているファイルからデータを読み込むときに `load` 関数を使用する。複数の配列を含むファイルを読み込むときも `load` 関数を使用するが、その際、複数個の配列はディクショナリに似たデータ構造でオブジェクトに代入される。そのオブジェクトの `.files` 属性には、配列の名前の一覧が保存されている。

```
# binary (.npy)

a = np.load('data.npy')
a
# array([1, 2, 3, 4, 5])

# binary (.npz)

x = np.load('objects.npz')
x.files
# ['a', 'b']

x['a']
# array([1, 2, 3, 4, 5])

x['b']
# array([0, 0, 1, 0, 1])
```

データの書き出し / テキスト

NumPy の配列データをファイルに保存するもう 1 つの方法は、配列データをテキストデータとして書き出す方法である。可読性はあるものの、桁数の多い小数などを正確に保存できないことがある。テキストファイルとして書き出すとき、`savetxt` 関数を使用する。この関数のオプションを指定することで、データ間の区切り文字や小数の有効桁数を指定することができる。

```
1.0000000000000000e+00  
2.0000000000000000e+00  
3.0000000000000000e+00  
4.0000000000000000e+00  
5.0000000000000000e+00
```

`.txt`

```
# text (.txt)  
  
a = np.array([1, 2, 3, 4, 5])  
  
np.savetxt('data.tsv', a)  
  
np.savetxt('data.csv', a,  
           fmt='%.18e',  
           delimiter=',')
```

データの読み込み / テキスト

テキストファイルを読み込むときに `loadtxt` 関数または `genfromtxt` 関数を使用する。`loadtxt` 関数を使用するとき、読み込み時にエラーが発生した場合、ファイルの区切り文字やヘッダーに合わせて、`loadtxt` 関数のオプションを調整することで対処できる。また、`loadtxt` 関数は欠損値を含むファイルを処理できないため、欠損値を含む場合、`genfromtxt` 関数を使用する。

```
# text (.txt)

a = np.loadtxt('data.txt')
a
# array([1., 2., 3., 4., 5.])
```

農学生命情報科学特論 I

3

○ NumPy

○ Pandas

💡 画像解析



アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

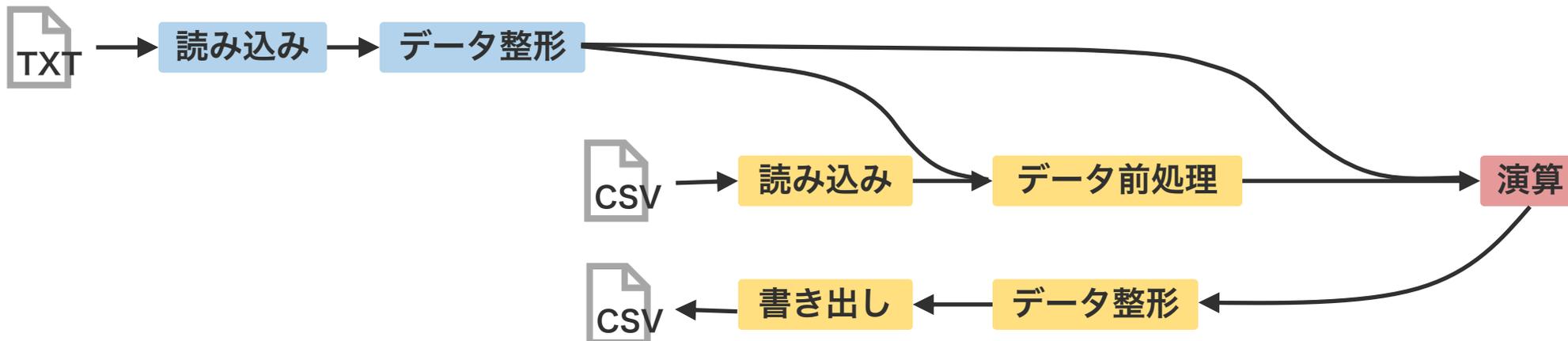
- リスト
- 2次元リスト

CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2次元配列



Pandas

- シリーズ
- データフレーム
- 表データ処理

シリーズ

Pandas のシリーズは、1 次元の配列データを扱うときに使用する。pd.Series 関数にリストを代入して作成する。シリーズから要素を取り出すときは、位置番号を指定して取り出す。

0	1	2	3	4	5
▼	▼	▼	▼	▼	▼
1	3	5	7	9	

```
import pandas as pd  
x = pd.Series([1, 3, 5, 7, 9])
```

```
x[2]  
# 5
```

シリーズ

シリーズは、リストと異なり、各値を位置番号と index の両方で管理している。シリーズを `pd.Series` 関数で作成するときに、index が自動的に作られるが、自ら指定して作ることもできる。

	0	1	2	3	4	5
index →	a	b	c	d	e	
	1	3	5	7	9	

シリーズの各要素に位置番号の他に index と呼ばれる索引が付けられる。

```
import pandas as pd
```

```
x = pd.Series([1, 3, 5, 7, 9])
```

```
x
```

```
# 0    1
```

```
# 1    3
```

```
# 2    5
```

```
# 3    7
```

```
# 4    9
```

```
# dtype: int64
```

```
x = pd.Series([1, 3, 5, 7, 9],  
              index=['a', 'b', 'c', 'd', 'e'])
```

```
x
```

```
# a    1
```

```
# b    3
```

```
# c    5
```

```
# d    7
```

```
# e    9
```

```
# dtype: int64
```

シリーズ

シリーズを作成するとき、文字列を index に指定した場合、その文字列でシリーズの各要素を取得できるようになる。

	0	1	2	3	4	5
index →	a	b	c	d	e	
	1	3	5	7	9	

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64

x[2]
# 5

x['c']
# 5
```

シリーズ

シリーズは、値と index の両方のデータを保持している。シリーズから値だけを NumPy の 1 次元配列として取得したい場合は、`x.values` のように取得する。また、シリーズから index だけを取得したい場合は、`x.index` を使用する。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x
# a      1
# b      3
# c      5
# d      7
# e      9
# dtype: int64

x.values
# array([1, 3, 5, 7, 9])

x.index
# Index(['a', 'b', 'c', 'd', 'e'],
#       dtype='object')
```

シリーズ

シリーズから要素を取得するとき、位置番号と index で取得できるほか、NumPy のようにスライスしたり、フィルター（ブーリアンベクトル）を使用して取得したりすることもできる。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
```

```
x[x < 5]
```

```
x[1:3]
```

```
k = ['b', 'c', 'e']
x[k]
```

シリーズ

$x < 5$

T	T	F	F	F
1	3	5	7	9

0 1 2 3 4 5
▼ ▼ ▼ ▼ ▼ ▼
a b c d e

1	3	5	7	9
---	---	---	---	---

0 1 2 3 4 5
▼ ▼ ▼ ▼ ▼ ▼
a b c d e

1	3	5	7	9
---	---	---	---	---

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
```

```
x[x < 5]
# a    1
# b    3
# dtype: int64
```

```
x[1:3]
# b    3
# c    5
# dtype: int64
```

```
k = ['b', 'c', 'e']
x[k]
# b    3
# c    5
# e    9
# dtype: int64
```

問題 P1-1

赤色で書かれているオブジェクトが保持している値を答えよ。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
```

```
x[['a', 'c', 'e']]
```

```
keep1 = (1 < x)
keep2 = (x < 7)
x[keep1 & keep2]
```

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
```

```
x['a'] = 2
```

```
x[0] = 4
```

```
x.values
```

```
x[x > 4] = 6
```

```
x.values
```

```
x[x % 2 == 0] = 1
```

```
x.values
```

シリーズ同士の計算

シリーズ同士は NumPy の配列と同様に四則演算が可能である。計算対象のシリーズの長さが同一でない場合は、見かけ上、短い方に欠損値を埋め込んだ上で計算される。また、シリーズとスカラーの計算も定義されている。この場合、スカラーが自動的に、シリーズと同じ長さに展開されて、計算が行われる（ブロードキャスト）。

```
import pandas as pd

x = pd.Series([1, 3, 5, 9])
y = pd.Series([2, 4, 6, 8])
z = pd.Series([1, 1])
w = 2

a = x + y
a.values
#array([ 3,  7, 11, 17])

b = x - z
b.values
# array([ 0.,  2., nan, nan])

c = x * w
c.values
# array([ 2,  6, 10, 18])
```

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

計算後の結果は index 順に並べ替えられる。x * y と y * x の計算結果は同じである。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['d', 'c', 'b', 'a'])

a = x + y
a.values
# array([9, 9, 9, 9])

x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'd', 'c'])

a = x * y
a.values
# array([ 2, 12, 40, 42])
```

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'd', 'e'])

y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'c', 'e'])

a = x + y
a.values
# array([ 3.,  7., nan, nan, 15.])

b = x * y
b.values
# array([ 2., 12., nan, nan, 56.])
```

要約統計量

Pandas のシリーズに対して、平均、分散、中央値などの要約統計量を計算するメソッドが多く用意されている。

```
import numpy as np
import pandas as pd
x = pd.Series([1, 2, 3, 4, 5])
```

```
x.count()
# 5
```

```
x.min()
# 1
```

```
x.max()
# 5
```

```
x.idxmax()
# 4
```

```
x.quantile(0.25)
# 2.0
```

```
x.sum()
# 15
```

```
x.mean()
# 3.0
```

```
x.median()
# 3.0
```

```
x.var()
# 2.5
```

```
x.std()
# 1.5811388300841898
```

```
x.cumsum().values
# array([ 1,  3,  6, 10, 15])
```

欠損値 / dropna

Pandas のシリーズに欠損値・非数値 (np.nan) を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

欠損値除去後のシリーズの要素数が変わるので、複数のシリーズを同時に解析するとき、要素数の違いによりブロードキャスト機能が働き、想定外の計算が行われる可能性があることに注意。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])
x
# a      3.0
# b      7.0
# c      NaN
# d      NaN
# e     15.0
# dtype: float64

y = x.dropna()
y
# 0  1.0
# 1  3.0
# 3  7.0
# dtype: float64
```

欠損値 / fillna

Pandas のシリーズに欠損値・非数値 (np.nan) を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

 合理的な根拠なしに、すべての欠損値を特定の値に置き換えてはならない。fillna メソッドを使用するときは十分に注意すること。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])

y = x.fillna(0)
y
# 0 1.0
# 1 3.0
# 2 0.0
# 3 7.0
# 4 0.0
# dtype: float64
```

欠損値 / isnull

Pandas のシリーズに欠損値・非数値 (np.nan) を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
<code>dropna</code>	シリーズ中の np.nan を取り除く。
<code>fillna</code>	シリーズ中の np.nan を指定した値で置き換える。
<code>isnull</code>	シリーズ中の各要素が np.nan かどうかを True と False で表す。
<code>notnull</code>	<code>isnull</code> と反対の動作を行う。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])

y = x.isnull()
y
# 0 False
# 1 False
# 2 True
# 3 False
# 4 True
# dtype: bool
```

欠損値 / notnull

Pandas のシリーズに欠損値・非数値 (np.nan) を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
<code>dropna</code>	シリーズ中の np.nan を取り除く。
<code>fillna</code>	シリーズ中の np.nan を指定した値で置き換える。
<code>isnull</code>	シリーズ中の各要素が np.nan かどうかを True と False で表す。
<code>notnull</code>	is.null と反対の動作を行う。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])

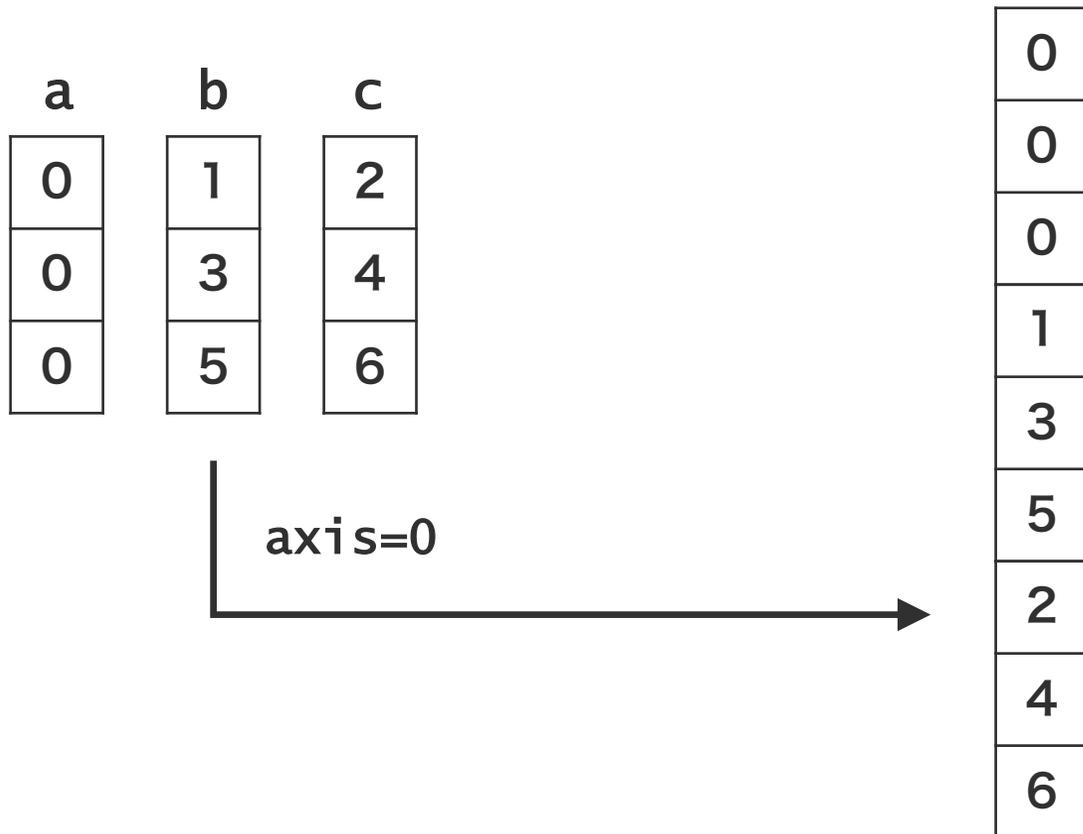
y = x.notnull()
y
# 0 True
# 1 True
# 2 False
# 3 True
# 4 False
# dtype: bool
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

データフレーム / pd.concat

Pandas では、行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることで作成される。シリーズ同士を束ねるとき `pd.concat` 関数を使用する。



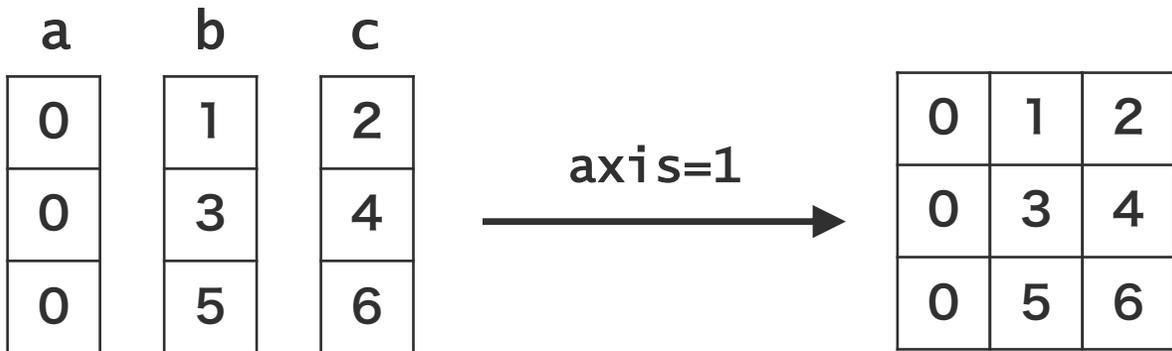
```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

x = pd.concat([a, b, c])
x
# 0      0
# 1      0
# 2      0
# 0      1
# 1      3
# 2      5
# 0      2
# 1      4
# 2      6
# dtype: int64

x.__class__.__name__
# 'Series'
```

データフレーム / pd.concat

pd.concat 関数の axis 引数を指定することで、複数のシリーズを列方向に束ねることもできる。



```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

y = pd.concat([a, b, c], axis=1)
y
#      0  1  2
# 0  0  1  2
# 1  0  3  4
# 2  0  5  6

y.__class__.__name__
# 'DataFrame'
```

データフレーム / ディクショナリ

リストを値として保存しているディクショナリを、Pandas のデータフレームに変換することもできる。このとき、ディクショナリのキーは、データフレームの列名に変換される。

```
import pandas as pd

d = {'buna' : [1, 0, 1, 0, 1],
     'kashi' : [1, 3, 5, 7, 9],
     'nara' : [0, 2, 4, 6, 8]}

df = pd.DataFrame(d)
df
#      buna  kashi  nara
# 0      1      1      0
# 1      0      3      2
# 2      1      5      4
# 3      0      7      6
# 4      1      9      8
```

データフレーム / ディクショナリ

シリーズを値として保存しているディクショナリも Pandas のデータフレームに変換することもできる。ディクショナリのキーは、データフレームの列名に変換される。また、シリーズの index は、データフレームの index（行名）に変換される。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
               index=['a', 'b', 'c', 'd', 'e'])
w2 = pd.Series([1, 3, 5, 7, 9],
               index=['a', 'b', 'c', 'd', 'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
               index=['a', 'b', 'c', 'd', 'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
```

#	buna	kashi	nara
# a	1	1	0
# b	0	3	2
# c	1	5	4
# d	0	7	6
# e	1	9	8

データフレーム / ディクショナリ

index を含むシリーズの場合、データフレームはそれらの index に基づいて作られる。データフレームは index で並べ替えられるため、その順番は必ずしもシリーズの順番を反映しないことに注意。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
               index=['b', 'a', 'c', 'd', 'f'])
w2 = pd.Series([1, 3, 5, 7, 9],
               index=['a', 'b', 'c', 'f', 'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
               index=['c', 'b', 'a', 'd', 'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
```

#	buna	kashi	nara
# a	0.0	1.0	4.0
# b	1.0	3.0	2.0
# c	1.0	5.0	0.0
# d	0.0	NaN	6.0
# e	NaN	9.0	8.0
# f	1.0	7.0	NaN

データフレーム / 二次元リスト

二次元リストを作成し、それをデータフレームに変換することもできる。この際に、データフレームの列名（columns）と行名（index）が自動的に振られる。なお、データフレームを作成するときに、columns と index 引数を利用することで、列名と行名を自由に付けることができる。

```
import pandas as pd
d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]])

d
#      0  1  2  3
# 0  11  12  13  14
# 1  21  22  23  24
# 2  31  32  33  34

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
                  index=['R1', 'R2', 'R3'],
                  columns=['C1', 'C2', 'C3', 'C4'])

d
#      C1  C2  C3  C4
# R1  11  12  13  14
# R2  21  22  23  24
# R3  31  32  33  34
```

データフレーム / 行名と列名

データフレームの `index` と `columns` は、あとから変更することができる。データフレームの `index` と `columns` 属性に直接新しい名前を代入することで変更できる。

 特定の列または行の名前だけを変更したいとき、Pandas の `rename` メソッドを使用すると便利である。

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html>

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
                  index=['R1', 'R2', 'R3'],
                  columns=['C1', 'C2', 'C3', 'C4'])
```

```
d
#      C1  C2  C3  C4
# R1   11  12  13  14
# R2   21  22  23  24
# R3   31  32  33  34
```

```
d.index = ['x', 'y', 'z']
d.columns = ['h', 'i', 'j', 'k']
```

```
d
#      h  i  j  k
# x   11  12  13  14
# y   21  22  23  24
# z   31  32  33  34
```

データフレーム要素参照 / iloc

データフレームから要素を取得する方法として、位置番号を利用しても、行名・列名を利用しても取得できる。位置番号で取得する場合は、iloc を使用し、0 から始まる位置番号を与える。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd
```

```
d = pd.DataFrame([[11, 12, 13, 14],  
                  [21, 22, 23, 24],  
                  [31, 32, 33, 34]],  
                  index=['R1', 'R2', 'R3'],  
                  columns=['C1', 'C2', 'C3', 'C4'])
```

```
d.iloc[0, :]  
# C1 11  
# C2 12  
# C3 13  
# C4 14
```

```
d.iloc[:, 2]  
# R1 13  
# R2 23  
# R3 33
```

データフレーム要素参照 / iloc

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd
```

```
d = pd.DataFrame([[11, 12, 13, 14],  
                  [21, 22, 23, 24],  
                  [31, 32, 33, 34]],  
                  index=['R1', 'R2', 'R3'],  
                  columns=['C1', 'C2', 'C3', 'C4'])
```

```
d.iloc[0:2, 1:4]  
#      C2  C3  C4  
# R1  12  13  14  
# R2  22  23  24
```

```
d.iloc[[0, 2], [1, 3]]  
#      C2  C4  
# R1  12  14  
# R3  32  34
```

NumPy の動作と異なるので、両者を混同しないように。

データフレーム要素参照 / loc

データフレームから要素を行名または列名で取得するときは、loc を使用する。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
                 index=['R1', 'R2', 'R3'],
                 columns=['C1', 'C2', 'C3', 'C4'])
```

```
d.loc['R1', :]
# C1 11
# C2 12
# C3 13
# C4 14
```

```
d.loc[:, 'C3']
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照 / loc

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd
```

```
d = pd.DataFrame([[11, 12, 13, 14],  
                  [21, 22, 23, 24],  
                  [31, 32, 33, 34]],  
                  index=['R1', 'R2', 'R3'],  
                  columns=['C1', 'C2', 'C3', 'C4'])
```

```
d.loc[['R1', 'R2'], 'C2':'C4']  
#      C2  C3  C4  
# R1  12  13  14  
# R2  22  23  24
```

```
d.loc[['R1', 'R3'], ['C2', 'C4']]  
#      C2  C4  
# R1  12  14  
# R3  32  34
```

データフレーム要素参照

データフレームもシリーズと同様に、True と False からなるフィルター（ブーリアンベクトル）を使って要素を取得することができる。フィルターを使用する場合は、loc または iloc の両方を使用することができる。

	C1	C2		keep		C1	C2	
R1	1	4	→	T	→	R1	1	4
R2	0	1		F		R3	1	0
R3	1	0		T		R4	1	3
R4	1	3		T				
R5	0	5		F				

```
import pandas as pd
```

```
d = pd.DataFrame([[1, 4],  
                  [0, 1],  
                  [1, 0],  
                  [1, 3],  
                  [0, 5]],  
                  index=['R1', 'R2', 'R3', 'R4', 'R5'],  
                  columns=['C1', 'C2'])
```

```
keep = (d.loc[:, 'C1'] > 0)
```

```
d.loc[keep, :]  
#      C1  C2  
# R1    1   4  
# R3    1   0  
# R4    1   3
```

前出の `.iloc` および `.loc` 以外にも、`.iat` および `.at` を利用した要素参照や列名・行名属性を用いた参照方法などがある。

```
import pandas as pd

df = pd.DataFrame([[1, 4], [0, 1],
                  [1, 0], [1, 3],
                  [0, 5]],
                 index=['R1', 'R2', 'R3', 'R4', 'R5'],
                 columns=['C1', 'C2'])

df.iloc[2:4, 0]
df.loc[['R1', 'R3'], ['C1']]
df.iat[0, 1]
df.at['R2', 'C2']
df[0:1]
df[['C1', 'C2']]
df.C2
```

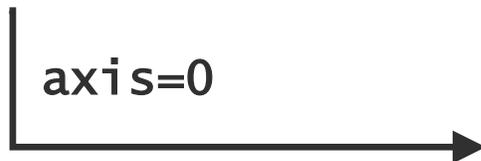
- `df.iloc` 整数からなる位置番号を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
- `df.loc` 列名あるいは行名を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
- `df.iat` `.iloc` と同じ使い方をする。ただし、1つの要素しか取得できない。
- `df.at` `.loc` と同じ使い方をする。ただし、1つの要素しか取得できない。
- `df[0:1]` 行の位置番号をスライス表記で指定して、該当行の要素を取得する。複数行の要素をまとめて取得できる。
- `df[['C1']]` 列の名前をリストで指定して、該当列の要素を取得する。複数列の要素をまとめて取得できる。ただし、スライス表記は使用できない。
- `df.C1` 列の名前をオブジェクトの属性として、該当列の要素を取得することができる。

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44



11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

```
import pandas as pd
```

```
d1 = pd.DataFrame([[11, 12, 13, 14],  
                  [21, 22, 23, 24]])
```

```
d2 = pd.DataFrame([[31, 32, 33, 34],  
                  [41, 42, 43, 44]])
```

```
df = pd.concat([d1, d2])
```

```
df
```

```
#      0  1  2  3  
# 0  11 12 13 14  
# 1  21 22 23 24  
# 2  31 32 33 34  
# 3  41 42 43 44
```

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14	31	32	33	34
21	22	23	24	41	42	43	44

↓ axis=1

11	12	13	14	31	32	33	34
21	22	23	24	41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2], axis=1)
df
#      0  1  2  3  0  1  2  3
# 0  11 12 13 14 31 32 33 34
# 1  21 22 23 24 41 42 43 44
```

データフレーム / pd.concat

データフレームに index または列名が存在するとき、データフレーム同士が index と列名に基づいて結合される。結合後のデータフレームの行と列の並び順に十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]],
                  index=['A', 'B'],
                  columns=['a', 'b', 'c', 'd'])

d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]],
                  index=['X', 'Y'],
                  columns=['a', 'b', 'c', 'e'])

df = pd.concat([d1, d2])
df
```

#		a	b	c	d	e
#	A	11	12	13	14.0	NaN
#	B	21	22	23	24.0	NaN
#	X	31	32	33	NaN	34.0
#	Y	41	42	43	NaN	44.0

データフレーム / pd.merge

複数のデータフレームを、特定の列の値に基づいて、マージすることが出来る。このとき `pd.merge` 関数を使用する。

k	V1		k	V2		k	V1	V2
a	1	inner merge	a	9		a	1	9
b	1		b	7		b	1	7
c	0		d	8				

データフレームの結合を行うとき、キーとなる列に重複要素が含まれると、予期しない挙動になる場合があるため、十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([[['a', 1],
                    ['b', 1],
                    ['c', 0]],
                  columns=['k', 'v1'])

d2 = pd.DataFrame([[['a', 9],
                    ['b', 7],
                    ['d', 8]],
                  columns=['k', 'v2'])

d = pd.merge(d1, d2) # how='inner'
d
#   k v1 v2
# 0 a  1  9
# 1 b  1  7
```

データフレーム / merge

k	V1		k	V2		k	V1	V2
a	1	outer merge →	a	9		a	1	9
b	1		b	7		b	1	7
c	0		c	0	NaN	c	0	NaN
			d	8		d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([[['a', 1],
                    ['b', 1],
                    ['c', 0]],
                  columns=['k', 'v1'])

d2 = pd.DataFrame([[['a', 9],
                    ['b', 7],
                    ['d', 8]],
                  columns=['k', 'v2'])

d = pd.merge(d1, d2, how='outer')
d
#      k    v1    v2
# 0    a    1.0    9.0
# 1    b    1.0    7.0
# 2    c    0.0   NaN
# 3    d   NaN    8.0
```

データフレーム / merge

k	V1		k	V2		k	V1	V2
a	1	left merge →	a	9		a	1	9
b	1		b	7		b	1	7
c	0		c	0	NaN			

```
import pandas as pd

d1 = pd.DataFrame([[['a', 1],
                    ['b', 1],
                    ['c', 0]],
                  columns=['k', 'v1'])

d2 = pd.DataFrame([[['a', 9],
                    ['b', 7],
                    ['d', 8]],
                  columns=['k', 'v2'])

d = pd.merge(d1, d2, how='left')
d
#      k  v1  v2
# 0  a    1  9.0
# 1  b    1  7.0
# 2  c    0  NaN
```

データフレーム / merge

k	V1		k	V2		k	V1	V2
a	1	right → merge	a	9		a	1	9
b	1		b	7		b	1	7
c	0		d	8		d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([[['a', 1],
                    ['b', 1],
                    ['c', 0]],
                  columns=['k', 'v1'])

d2 = pd.DataFrame([[['a', 9],
                    ['b', 7],
                    ['d', 8]],
                  columns=['k', 'v2'])

d = pd.merge(d1, d2, how='right')
d
#      k    v1  v2
# 0    a  1.0   9
# 1    b  1.0   7
# 2    d  NaN   8
```

データフレーム / merge

マージ対象のデータフレームの基準列の列名が異なる場合は、pd.merge 関数の left_on および right_on 引数で指定する。

k	V1	f	V2
a	1	a	9
b	1	b	7
c	0		

merge

k	V1	f	V2
a	1	a	9
b	1	b	7
c	0	NaN	NaN
NaN	NaN	d	8

```
import pandas as pd

d1 = pd.DataFrame([[['a', 1],
                    ['b', 1],
                    ['c', 0]],
                  columns=['k', 'v1'])

d2 = pd.DataFrame([[['a', 9],
                    ['b', 7],
                    ['d', 8]],
                  columns=['f', 'v2'])

d = pd.merge(d1, d2, how='outer',
             left_on='k', right_on='f')

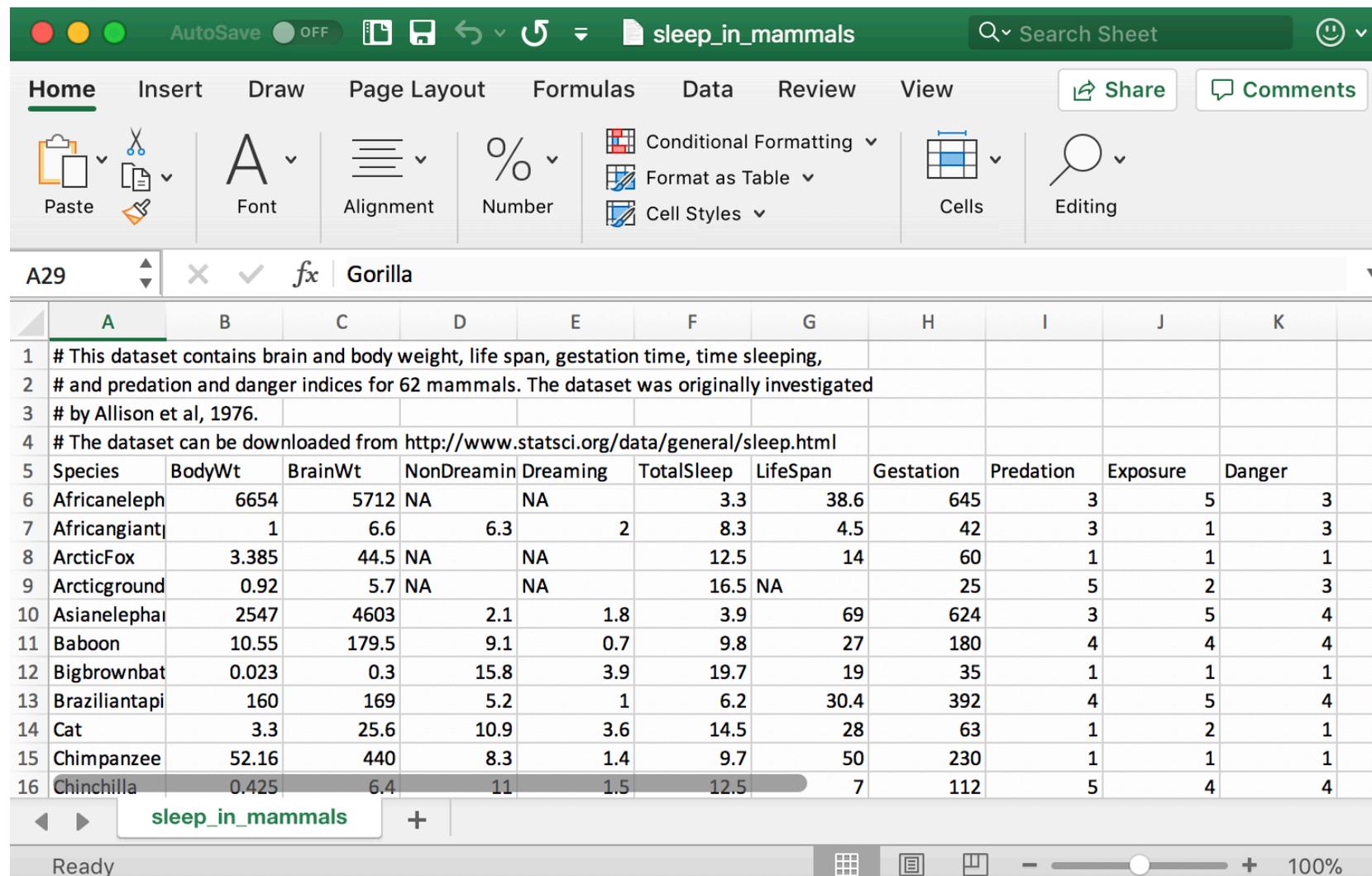
d
#      k  v1  f  v2
# 0    a  1.0  a  9.0
# 1    b  1.0  b  7.0
# 2    c  0.0 NaN NaN
# 3  NaN NaN  d  8.0
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

表データ

生物学で取り扱うデータは、一般的に、1 サンプル 1 行で記載されている。また、各行は複数の項目からなり、サンプルの属性が記載されている。このようなデータは、一般的にタブ区切りファイル (TSV) またはカンマ区切りファイル (CSV) のテキストファイルで保存される。



The screenshot shows a spreadsheet application window titled "sleep_in_mammals". The interface includes a menu bar with options like Home, Insert, Draw, Page Layout, Formulas, Data, Review, and View. Below the menu is a ribbon with various tool icons for formatting and editing. The spreadsheet itself contains a table with 16 rows of data. The first four rows are comments, and the remaining rows are data points for various mammal species. The columns represent different attributes: Species, BodyWt, BrainWt, NonDreamin, Dreaming, TotalSleep, LifeSpan, Gestation, Predation, Exposure, and Danger. The status bar at the bottom indicates "Ready" and a zoom level of 100%.

	A	B	C	D	E	F	G	H	I	J	K
1	# This dataset contains brain and body weight, life span, gestation time, time sleeping,										
2	# and predation and danger indices for 62 mammals. The dataset was originally investigated										
3	# by Allison et al, 1976.										
4	# The dataset can be downloaded from http://www.statsci.org/data/general/sleep.html										
5	Species	BodyWt	BrainWt	NonDreamin	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
6	Africaneleph	6654	5712	NA	NA	3.3	38.6	645	3	5	3
7	Africangiant	1	6.6	6.3	2	8.3	4.5	42	3	1	3
8	ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1	1	1
9	Arcticground	0.92	5.7	NA	NA	16.5	NA	25	5	2	3
10	Asianeleph	2547	4603	2.1	1.8	3.9	69	624	3	5	4
11	Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4	4	4
12	Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35	1	1	1
13	Braziliantapi	160	169	5.2	1	6.2	30.4	392	4	5	4
14	Cat	3.3	25.6	10.9	3.6	14.5	28	63	1	2	1
15	Chimpanzee	52.16	440	8.3	1.4	9.7	50	230	1	1	1
16	Chinchilla	0.425	6.4	11	1.5	12.5	7	112	5	4	4

表データ

コメント

```
# This dataset contains brain and body weight, life span, gestation time,  
# and predation and danger indices for 62 mammals. The dataset was origin  
# by Allison et al, 1976.
```

```
# The dataset can be downloaded from http://www.statsci.org/data/general/
```

データ

```
Species BodyWt BrainWt NonDreaming Dreaming TotalSleep L  
Africanelephant 6654 5712 NA NA 3.3 38.6 645 3  
Africangiantpouchedrat 1 6.6 6.3 2 8.3 4.5 4  
ArcticFox 3.385 44.5 NA NA 12.5 14 60 1  
Arcticgroundsquirrel 0.92 5.7 NA NA 16.5 NA 2  
Asianelephant 2547 4603 2.1 1.8 3.9 69 624 3  
Baboon 10.55 179.5 9.1 0.7 9.8 27 180 4 4  
Bigbrownbat 0.023 0.3 15.8 3.9 19.7 19 35 1  
Braziliantapir 160 169 5.2 1 6.2 30.4 392 4  
Cat 3.3 25.6 10.9 3.6 14.5 28 63 1 2  
Chimpanzee 52.16 440 8.3 1.4 9.7 50 230 1  
Chinchilla 0.425 6.4 11 1.5 12.5 7 112 5  
Cow 465 423 3.2 0.7 3.9 30 281 5 5
```

表データ

属性 (特徴量)

ヘッダー	Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	L			
サンプル	Africanelephant	6654	5712	NA	NA	3.3	38.6	645	3	
	Africangiantpouchedrat	1	欠損値	6.6	6.3	2	8.3	4.5	4	
	ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1	
	Arcticgroundsquirrel	0.92	5.7	NA	NA	16.5	NA	2	2	
	Asianelephant	2547	4603	2.1	1.8	3.9	69	624	3	
	Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4	4
	Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35	1	
	Braziliantapir	160	169	5.2	1	6.2	30.4	392	4	
	Cat	3.3	25.6	10.9	3.6	14.5	28	63	1	2
	Chimpanzee	52.16	440	8.3	1.4	9.7	50	230	1	
	Chinchilla	0.425	6.4	11	1.5	12.5	7	112	5	
	Cow	465	423	3.2	0.7	3.9	30	281	5	5

表データ読み込み

Pandas の `read_table` 関数を使うことで、CSV または TSV データを読み込むことができる。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f)
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()
pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()
ParserError: Error tokenizing data. C error: Expected 1 fields in line 5, saw 11
```



`sleep_in_mammals.txt` ファイルは、`read_table` 関数が想定しているフォーマットに従っていないため、エラーになった。

表データ読み込み

Pandas の `read_table` 関数を使うことで、CSV または TSV データを読み込むことができる。データを正しく読み込むには、`read_table` 関数に、コメント行を明示し、ヘッダ行の有無、区切り文字の種類を正しく指定する必要がある。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f,
                comment='#',
                header=0,
                sep='\t')
```



バックslashは特別な意味を持つ文字である。バックslashの後に続く1文字は、特別な意味を持つ。例えば 't' は英文字の t を表すが、'\t' はタブを表す。

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t')

d.shape
# (62, 11)

d.head()
#           Species      Bodywt  Brainwt  ...  Predation  Exposure  Danger
# 0      Africanelephant  6654.000   5712.0  ...         3         5         3
# 1  Africangiantpouchedrat    1.000     6.6  ...         3         1         3
# 2           ArcticFox    3.385    44.5  ...         1         1         1
# 3  Arcticgroundsquirrel    0.920     5.7  ...         5         2         3
# 4           Asianelephant  2547.000   4603.0  ...         3         5         4
# [5 rows x 11 columns]
```

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

特定の列をデータの一部ではなくて、行名として取り組むこともできる。

```
d.shape
# (62, 10)
```

```
d.head()
```

```
#           Bodywt  Brainwt  ...  Exposure  Danger
# Species
# Africanelephant    6654.000    5712.0  ...         5         3
# Africangiantpouchedrat    1.000         6.6  ...         1         3
# ArcticFox          3.385        44.5  ...         1         1
# Arcticgroundsquirrel    0.920         5.7  ...         2         3
# Asianelephant     2547.000    4603.0  ...         5         4
# [5 rows x 10 columns]
```

表データ / 行名と列名

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.index
# Index(['Africanelephant', 'Africangiantpouchedrat', 'ArcticFox',
#        'Arcticgroundsquirrel', 'Asianelephant', 'Baboon', 'Bigbrownbat',
#        'Braziliantapir', 'Cat', 'Chimpanzee', 'Chinchilla', 'Cow',
#        ...,
#        'Treeshrew', 'Vervet', 'Wateropossum', 'Yellow-belliedmarmot'],
#        dtype='object', name='Species')

d.columns
# Index(['Bodywt', 'Brainwt', 'NonDreaming', 'Dreaming', 'TotalSleep',
#        'LifeSpan', 'Gestation', 'Predation', 'Exposure', 'Danger'],
#        dtype='object')
```

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.iloc[0:3, 0:5]
#           Bodywt  Brainwt  NonDreaming  Dreaming  TotalSleep
# Species
# Africanelephant    6654.000    5712.0           NaN           NaN           3.3
# Africangiantpouchedrat    1.000         6.6           6.3           2.0           8.3
# ArcticFox          3.385         44.5           NaN           NaN           12.5

d.iloc[0:2, :]
#           Bodywt  Brainwt  NonDreaming  ...  Exposure  Danger
# Species
# Africanelephant    6654.000    5712.0           NaN  ...           5           3
# Africangiantpouchedrat    1.000         6.6           6.3  ...           1           3
```

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

species = ['Cat', 'Rat', 'Cow', 'Pig']
features = ['Bodywt', 'Brainwt', 'TotalSleep', 'LifeSpan']
```

```
d.loc[species, features]
```

#	Bodywt	Brainwt	TotalSleep	LifeSpan
# Species				
# Cat	3.30	25.6	14.5	28.0
# Rat	0.28	1.9	13.2	4.7
# Cow	465.00	423.0	3.9	30.0
# Pig	192.00	180.0	8.4	27.0

表データ読み込み / データ要約

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

```
d.describe()
```

```
#           BodyWt      BrainWt  NonDreaming  ...  Predation  Exposure      Danger
# count      62.000000     62.000000     48.000000  ...    62.000000    62.000000    62.000000
# mean      198.789984     283.134194      8.672917  ...     2.870968     2.419355     2.612903
# std       899.158011     930.278942      3.666452  ...     1.476414     1.604792     1.441252
# min         0.005000      0.140000      2.100000  ...     1.000000     1.000000     1.000000
# 25%         0.600000      4.250000      6.250000  ...     2.000000     1.000000     1.000000
# 50%         3.342500     17.250000      8.350000  ...     3.000000     2.000000     2.000000
# 75%        48.202500    166.000000     11.000000  ...     4.000000     4.000000     4.000000
# max       6654.000000   5712.000000     17.900000  ...     5.000000     5.000000     5.000000
# [8 rows x 10 columns]
```

表データ読み込み / データ可視化

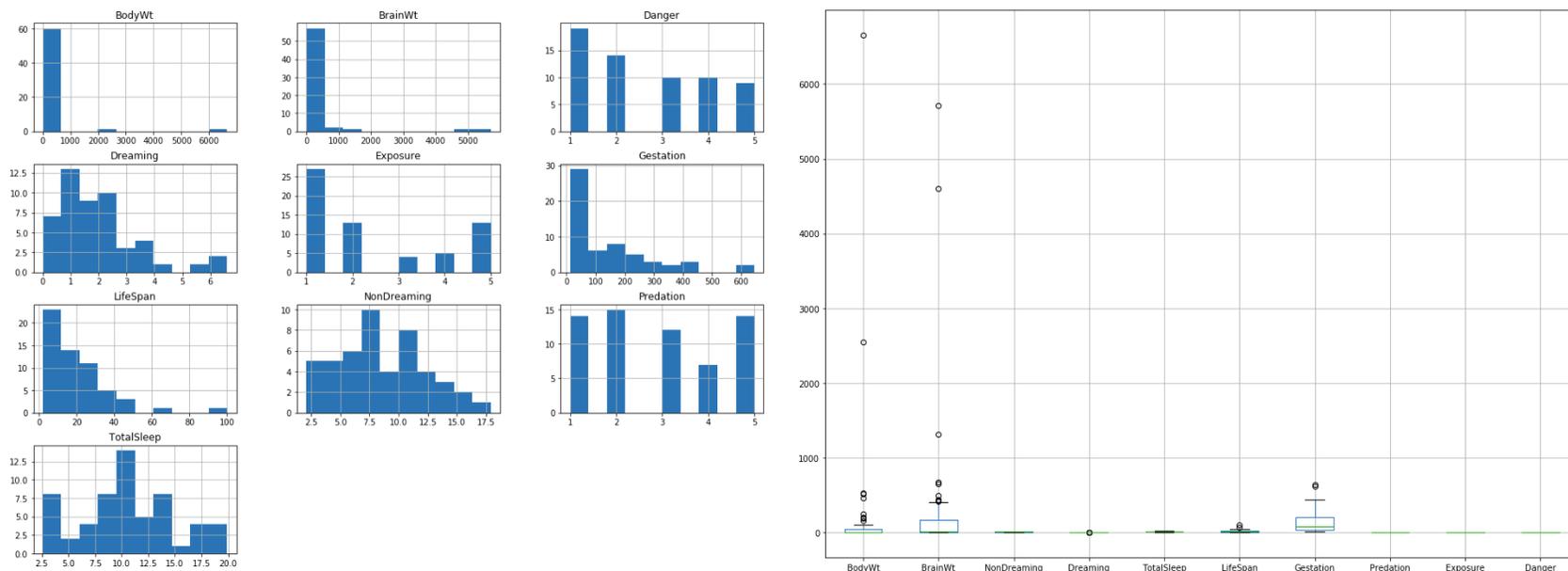
```
import pandas as pd
```

```
f = 'sleep_in_mammals.txt'
```

```
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

```
d.hist()
```

```
d.boxplot()
```



※ 一部の属性の値の範囲が大きいため、対数化してからグラフに描くと全体の傾向を掴めやすくなる。

※ matplotlib でグラフを描いた方が一般的であるので、ここでは Pandas の視覚化機能を取り上げない。

問題 P2-1

diversity_galapagos.txt を Pandas で読み込み、面積 (Area) が最も大きい島の名前とその面積を答えよ。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```



ヒント

シリーズ s の最大値は $\max(s)$ で求めることができる。

 https://aabdd.jp/notes/data/diversity_galapagos.txt

問題 P2-2

diversity_galapagos.txt を Pandas で読み込み、各島における面積あたりの種数を求めよ。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

問題 P2-3

trees.txt を Pandas で読み込み、高さ (Height) 80インチ以上の木の外周長 (Girth) の平均値と分散、高さ 80インチ未満の木の外周長の平均値と分散を求めよ。なお、平均値と分散を求める関数はそれぞれ mean および var である。

```
import pandas as pd  
  
f = 'trees.txt'
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、`to_csv` 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
               index_col=0)

d.to_csv('o.txt', sep=',', header=False, index=False)
```

```
6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
3.385,44.5,,,12.5,14.0,60.0,1,1,1
0.92,5.7,,,16.5,,25.0,5,2,3
2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
52.16,440.0,8.3,1.4,9.7,50.0,230.0,1,1,1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、`to_csv` 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
               index_col=0)

d.to_csv('o.txt', sep=',', header=True, index=True)
```

```
-----
Species,BodyWt,BrainWt,NonDreaming,Dreaming,TotalSleep,LifeSpan,Gestation,Predation,Exposure
Africanelephant,6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
Africangiantpouchedrat,1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
ArcticFox,3.385,44.5,,,12.5,14.0,60.0,1,1,1
Arcticgroundsquirrel,0.92,5.7,,,16.5,,25.0,5,2,3
Asianelephant,2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
Baboon,10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
Bigbrownbat,0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
Braziliantapir,160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
Cat,3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
-----
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、`to_csv` 関数 (メソッド) を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
               index_col=0)

d.to_csv('o.txt', sep='\t', header=True, index=True)
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestati
Africanelephant	6654.0	5712.0		3.3	38.6	645.0	3
Africangiantpouchedrat		1.0	6.6	6.3	2.0	8.3	42.0
ArcticFox	3.385	44.5		12.5	14.0	60.0	1
Arcticgroundsquirrel		0.92	5.7		16.5	25.0	5
Asianelephant	2547.0	4603.0	2.1	1.8	3.9	69.0	624.0
Baboon	10.55	179.5	9.1	0.7	9.8	27.0	180.0
Bigbrownbat		0.023	0.3	15.8	3.9	19.7	19.0
Braziliantapir	160.0	169.0	5.2	1.0	6.2	30.4	392.0
Cat	3.3	25.6	10.9	3.6	14.5	28.0	63.0

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、`to_csv` 関数 (メソッド) を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
               index_col=0)

d.to_csv('o.txt', sep='\t', header=True, index=True)
na_rep= 'NA')
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestati			
Africanelephant	6654.0	5712.0	NA	NA	3.3	38.6	645.0	3	5	3
Africangiantpouchedrat	1.0	6.6	6.3	2.0	8.3	4.5	42.0	3	1	1
ArcticFox	3.385	44.5	NA	NA	11.5	6	1	1	1	1
Arcticgroundsquirrel	0.92	5.7	NA	NA	16.5	NA	25.0	5	2	2
Asianelephant	2547.0	4603.0	2.1	1.8	3.9	69.0	624.0	3	5	4
Baboon	10.55	179.5	9.1	0.7	9.8	27.0	180.0	4	4	4
Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19.0	35.0	1	1	1
Braziliantapir	160.0	169.0	5.2	1.0	6.2	30.4	392.0	4	5	4
Cat	3.3	25.6	10.9	3.6	14.5	28.0	63.0	1	2	1

空白がNAに置換される

グループ演算

Pandas のデータフレームでは、ある列の値に基づいて、全データをいくつかのサブセット分け、それぞれのサブセットに対して平均や分散などを計算する機能が提供されている。



iris データセットは、3種のアヤメ (setosa・versicolor・virginica) に対して、萼 sepal の長さ length と幅 width、花弁 petal の長さ length と幅 width を測定したデータである。各種には 50 個体のデータ含まれ、3種で計150個体のデータが含まれている。

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

```
d.head()
```

#	ID	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
# 0	1	5.1	3.5	1.4	0.2	setosa
# 1	2	4.9	3.0	1.4	0.2	setosa
# 2	3	4.7	3.2	1.3	0.2	setosa
# 3	4	4.6	3.1	1.5	0.2	setosa
# 4	5	5.0	3.6	1.4	0.2	setosa



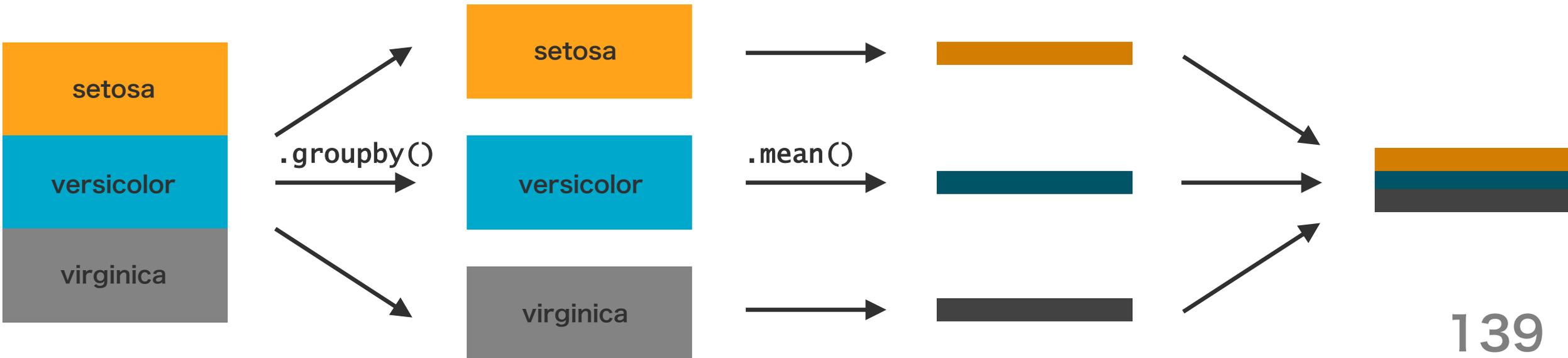
↓ <https://aabbdd.jp/notes/data/iris.txt>

グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

```
d.groupby('Species').mean()
```

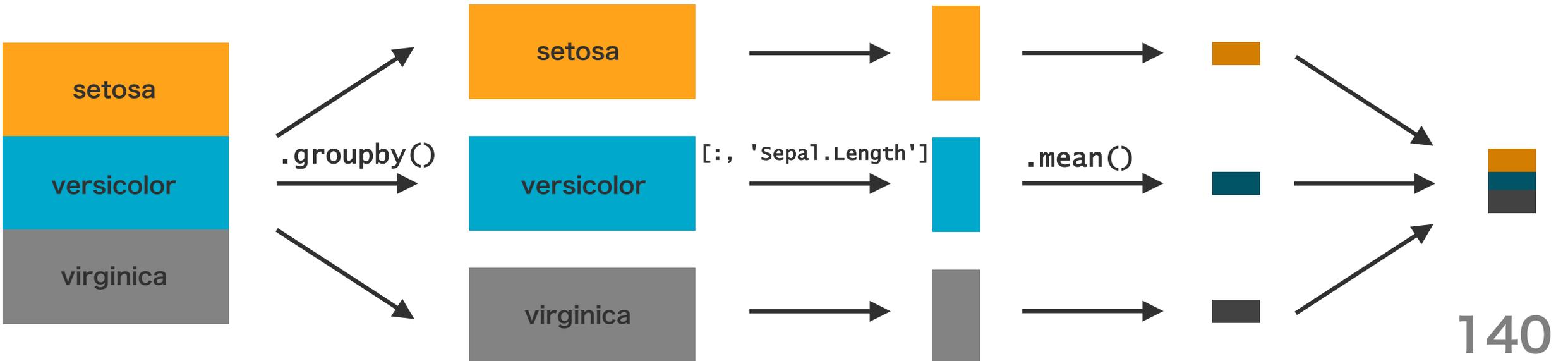
```
#           ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# Species
# setosa      25.5          5.006          3.428          1.462          0.246
# versicolor  75.5          5.936          2.770          4.260          1.326
# virginica  125.5          6.588          2.974          5.552          2.026
```



グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.groupby('Species').loc[:, 'Sepal.Length'].mean()
# Species
# setosa      5.006
# versicolor  5.936
# virginica   6.588
# Name: Sepal.Length, dtype: float64
```



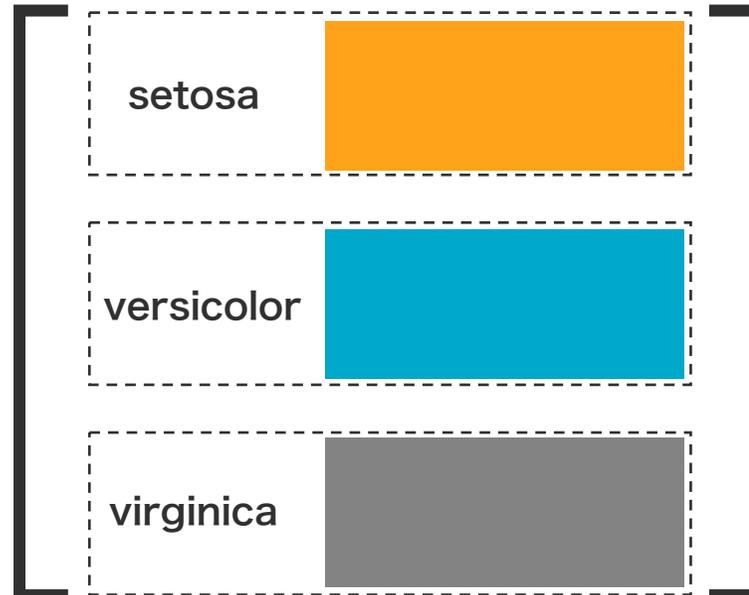
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

for gname, subset in d.groupby('Species'):
    print(gname)
    print(subset.head())
```



.groupby()



要素が 3 個あるリストとして for 構文処理できる。



for 構文を利用してリストの各要素を処理

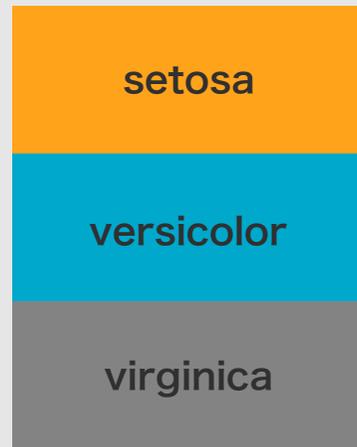
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

```
for gname, subset in d.groupby('Species'):
    M = subset.iloc[:, 1:5].max(axis=0)
    m = subset.iloc[:, 1:5].min(axis=0)
    print(gname)
    print(M - m)
```

```
# setosa
# Sepal.Length 1.5
# Sepal.Width 2.1
# Petal.Length 0.9
# Petal.Width 0.5
# versicolor
# Sepal.Length 2.1
# Sepal.Width 1.4
# ...
# ...
```

2列目から6列目の部分列を抽出し、各列の最大値を計算する。



.groupby()



グループ演算 / apply

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

```
def calc_diff(df):
    M = df.iloc[:, 1:5].max(axis=0)
    m = df.iloc[:, 1:5].min(axis=0)
    return M - m
```

for 構文を利用したグループ演算を関数化することで、高速な apply 関数を利用できるようなる。

```
d.groupby('Species').apply(calc_diff)
#           Sepal.Length  Sepal.width  Petal.Length  Petal.width
# Species
# setosa                1.5           2.1           0.9           0.5
# versicolor            2.1           1.4           2.1           0.8
# virginica             3.0           1.6           2.4           1.1
```

問題 P3-1

iris.txtを読み込み、各種におけるにおける、花弁 petal の長さ length と幅 width の比率の平均値を求めよ。

```
import pandas as pd

f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

問題 P3-2

rice.txt には、野生型イネ wt と実験処理後のイネ ANU842 を 3 つの環境 (F10, NH4Cl, NH4NO3) で栽培し、その収穫後のバイオマスの乾燥重量 (shoot_dry_mass, root_dry_mass) を測定したデータが記録されている。F10 栽培環境で栽培したイネ wt の shoot_dry_mass と root_dry_mass の平均値をそれぞれ求めよ。

```
import pandas as pd
```

```
f = 'rice.txt'
```

```
d = pd.read_csv(f, header=0, sep='\t')
```

```
d.head()
```

#	replicate	block	root_dry_mass	shoot_dry_mass	trt	fert	variety
# 0	1	1	56	132	F10	F10	wt
# 1	2	1	66	120	F10	F10	wt
# 2	3	1	40	108	F10	F10	wt
# 3	4	1	43	134	F10	F10	wt
# 4	5	1	55	119	F10	F10	wt

農学生命情報科学特論 I

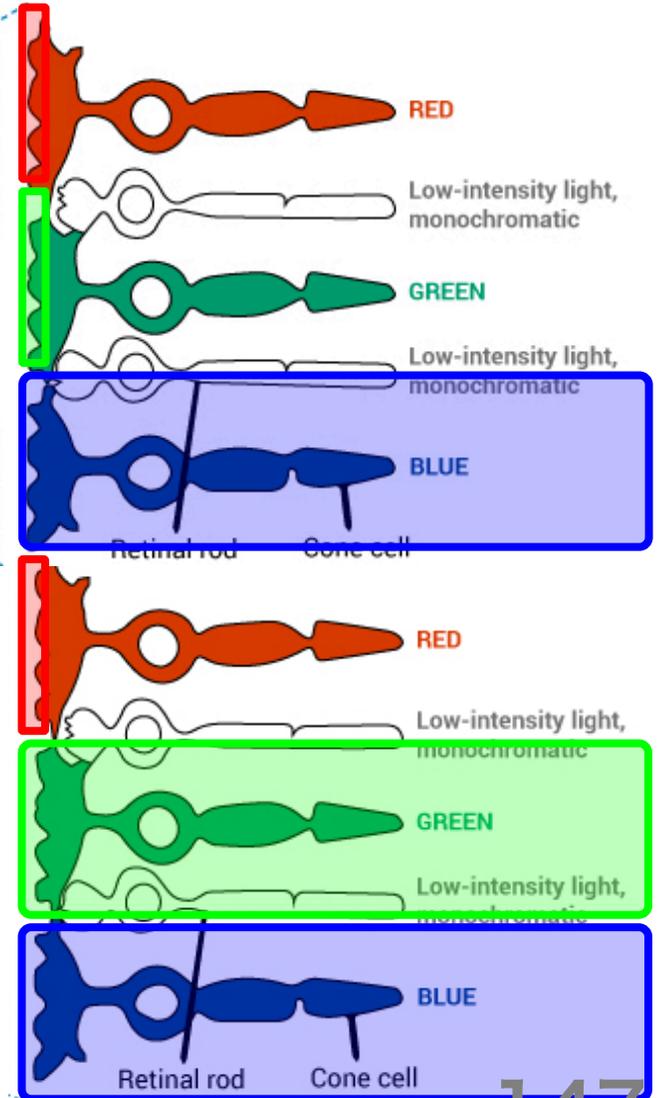
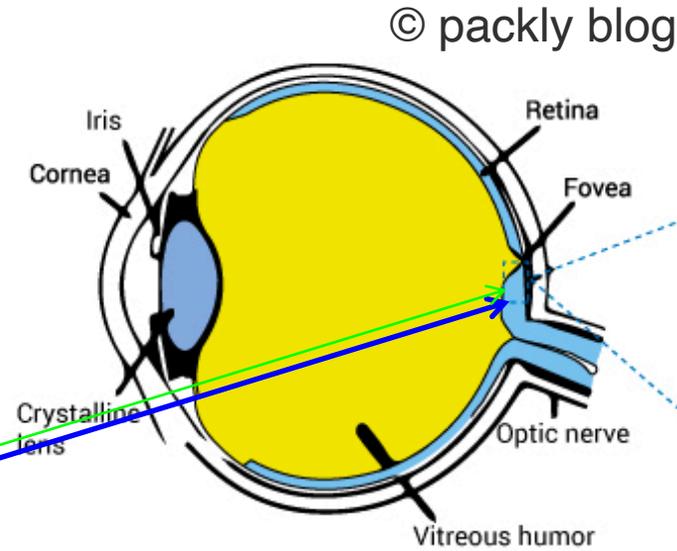
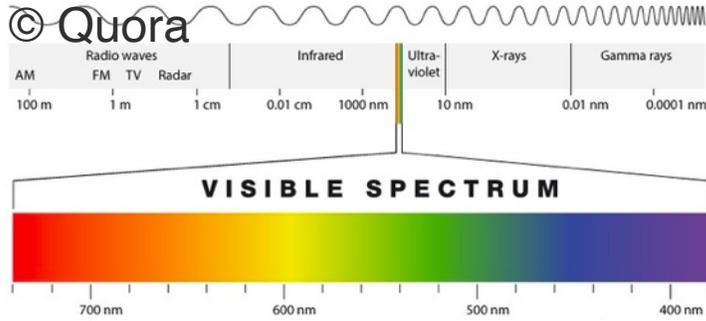
3

NumPy

Pandas

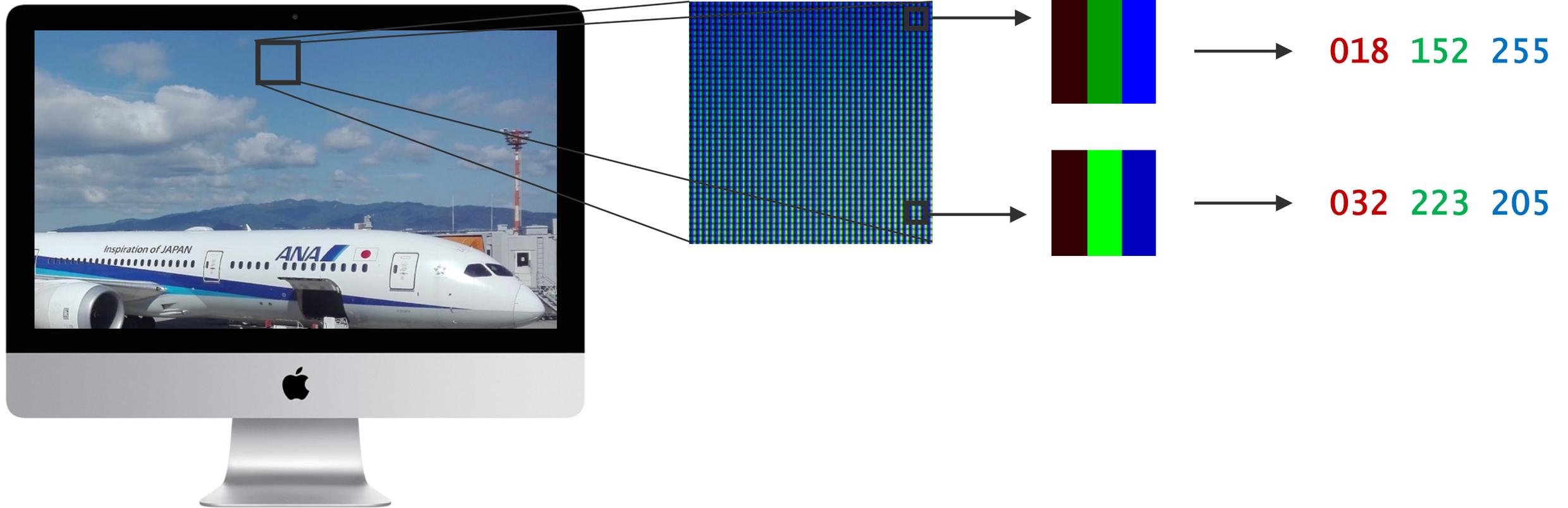
画像解析

物体認識

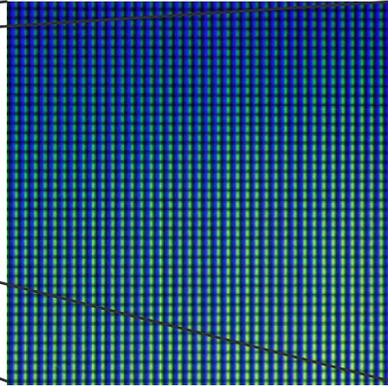


青色反射光が強いので、ほぼすべての青色桿体細胞の神経が励起する。緑色反射光がそれほど強くないので、一部だけの緑色桿体細胞が励起する。赤色反射光がほとんどないので、赤色反対細胞がほとんど励起しない。脳は、励起する桿体細胞の割合を総合判断し、色として認識する。

物体認識



物体認識



021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

物体分類

物体分類

特徴量



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024
```

```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

```
221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

特徴抽出

```
102
074
144
221
002
140
211
120
104
⋮
⋮
```

分類

飛行機

深層学習



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024
```

```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

```
221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

CNN

飛行機

特徴抽出



021 031 022 063 042
 041 065 034 044 033
 082 024 056 072 038
 046 046 033 049 029
 025 061 012 103 052
 031 051 053 081 041
 093 049 058 092 036
 054 101 061 024 041
 109 023 020 052 022
 111 042 024 021 024

162 149 144 131 124
 163 202 123 101 143
 132 121 146 150 142
 178 178 183 129 126
 144 125 135 112 171
 241 201 191 122 120
 132 152 152 137 121
 145 150 171 104 143
 152 151 154 114 141
 151 149 123 101 153

221 220 224 230 224
 223 209 230 231 243
 223 229 240 220 217
 220 221 217 230 236
 244 250 251 221 216
 252 251 250 212 220
 245 251 251 247 231
 253 249 249 254 221
 251 254 255 245 253
 254 253 250 249 251

特徴抽出

SIFT HoG textons RIFT GLOH

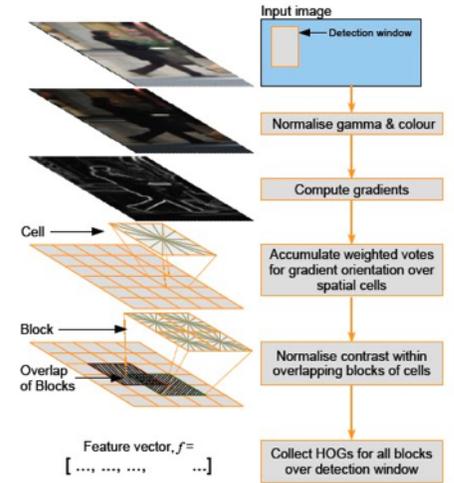
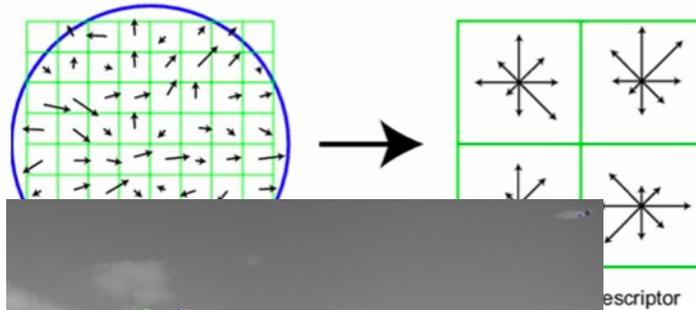


Fig. 3.3. An overview of static HOG feature extraction. The detector window is tiled with a grid of overlapping blocks. Each block contains a grid of spatial cells. For each cell, the weighted vote of image gradients in orientation histograms is performed. These are locally normalised and collected in one big feature vector.

特徴抽出



021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024

162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153

221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251



特徴抽出

102
074
144
221
002
211
104
085
101
113
032
111
046
210
142
228
107
123
192
126
114
125
135
171
201

特徴抽出



025 061 012 103 052
021 031 022 163 142
182 024 011 072 038
193 149 158 092 136
141 165 134 244 133
246 146 233 249 129
131 251 253 211 241
211 242 224 221 224
254 201 241 224 241
209 223 220 252 222

178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
151 149 123 101 153
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141

152 151 250 212 220
151 054 255 245 253
121 020 224 230 224
020 021 217 230 236
023 109 230 231 243
023 029 040 020 017
044 050 051 121 016
054 053 050 149 151
045 051 151 147 131
053 009 149 154 121



102 002
074 103
144 211
221 104
002 085
211 101
104 113
085 032
101 144
113 221
032 052
111 111
046 146
210 104
142 142
228 228
107 107
123 123
192 015
126 135
114 211
125 201
135 192
171 162
201 141



特徴抽出

特徴抽出



021 031 022 063 042
041 065 034 044 033
025 061 212 103 052
082 224 252 212 038
111 242 224 221 024
046 246 133 249 029
054 201 211 224 041
109 023 220 152 022
031 051 053 181 041
093 049 058 092 136

244 225 235 212 211
241 201 191 222 220
252 251 254 214 241
151 049 023 001 153
162 049 044 031 224
213 102 023 001 243
232 021 046 050 242
178 018 183 029 226
232 012 152 237 221
245 050 211 204 243

152 051 150 212 120
145 151 051 047 131
154 153 050 049 151
053 149 049 054 021
120 021 017 030 236
044 150 151 121 116
021 120 024 130 124
123 109 030 131 143
023 129 040 120 117
051 024 025 045 053



102 002 141
074 103 146
144 211 104
221 104 142
002 085 002
211 101 103
104 113 211
085 032 144
101 144 221
113 221 052
032 052 228
111 111 107
046 146 123
210 104 015
142 142 135
228 228 211
107 107 201
123 123 192
192 015 162
126 135 141
114 211 104
125 201 085
135 192 101
171 162 113
201 141 032



特徴抽出

物体分類



021 031 022 063 042
041 065 034 044 033
025 061 212 103 052
082 224 252 212 038
111 242 224 221 024
046 246 133 249 029
054 201 211 224 041
109 023 220 152 022
031 051 053 181 041
093 049 058 092 136

244 225 235 212 211
241 201 191 222 220
252 251 254 214 241
151 049 023 001 153
162 049 044 031 224
213 102 023 001 243
232 021 046 050 242
178 018 183 029 226
232 012 152 237 221
245 050 211 204 243

152 051 150 212 120
145 151 051 047 131
154 153 050 049 151
053 149 049 054 021
120 021 017 030 236
044 150 151 121 116
021 120 024 130 124
123 109 030 131 143
023 129 040 120 117
051 024 025 045 053



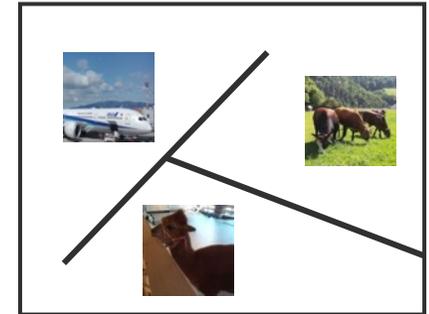
特徴抽出



102	002	141
074	103	146
144	211	104
221	104	142
002	085	002
211	101	103
104	113	211
085	032	144
101	144	221
113	221	052
032	052	228
111	111	107
046	146	123
210	104	015
142	142	135
228	228	211
107	107	201
123	123	192
192	015	162
126	135	141
114	211	104
125	201	085
135	192	101
171	162	113
201	141	032



機械学習



物体分類

物体分類

特徴量



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024
```

```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

```
221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

特徴抽出

102
074
144
221
002
140
211
120
104
:
:

分類

飛行機

深層学習



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024
```

```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

```
221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

CNN

飛行機

畳み込みニューラルネットワーク

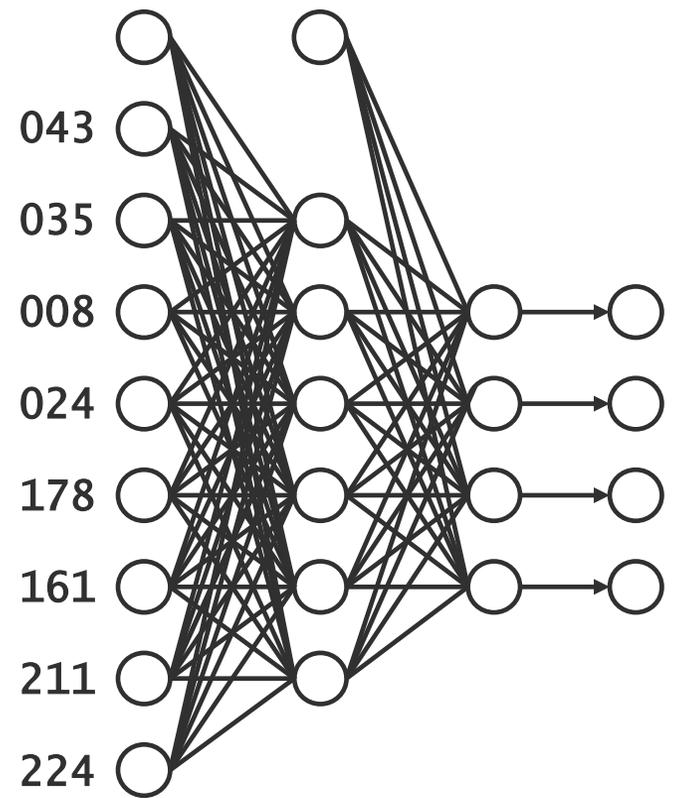
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029 224
025 061 032 030 052
031 051 053 081 041 217
093 049 058 091 036 236
054 101 061 024 104 216
109 023 020 052 022 220
111 042 024 021 024 231
151 149 123 101 153
251 254 255 245 253
254 253 250 249 251



021 031 022
041 065 034 251
082 024 056 250 034
049 029 02 225 103 1023
012 103 05 23 5
194 231 19 208 024 095
210 231 05 02 115 124
019 101 03 115 7201
223 103 221
115 198 108



043 035
008 024
178 161
211 224



畳み込み
(特徴抽出)

ニューラルネットワーク
(分類)

Grayscale Photo



162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

1 channel

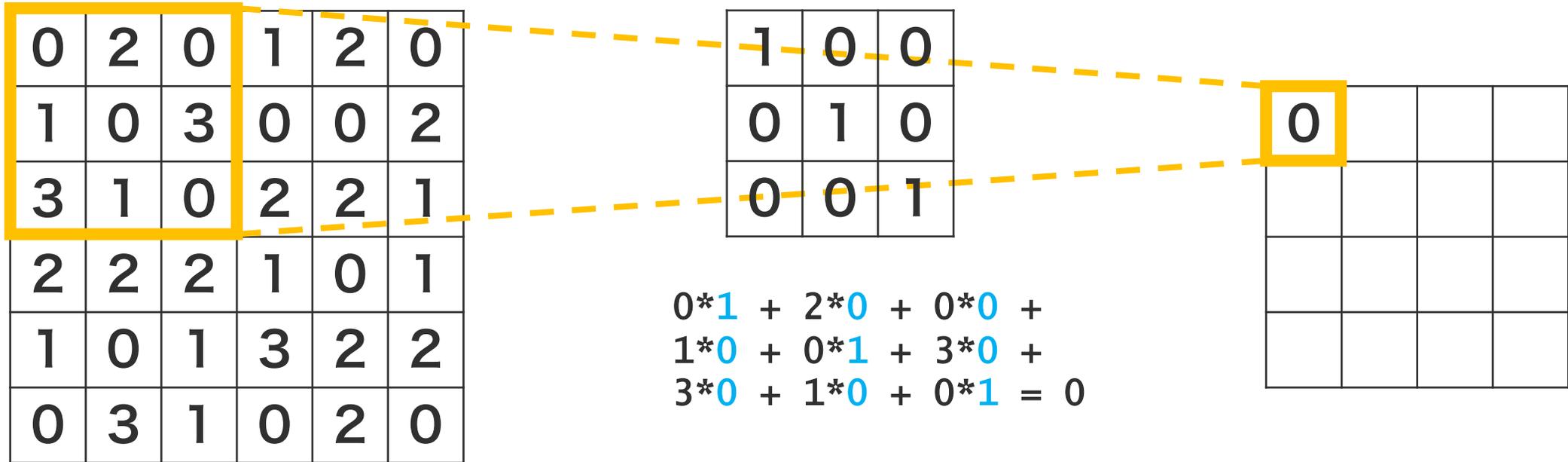
Color Photo



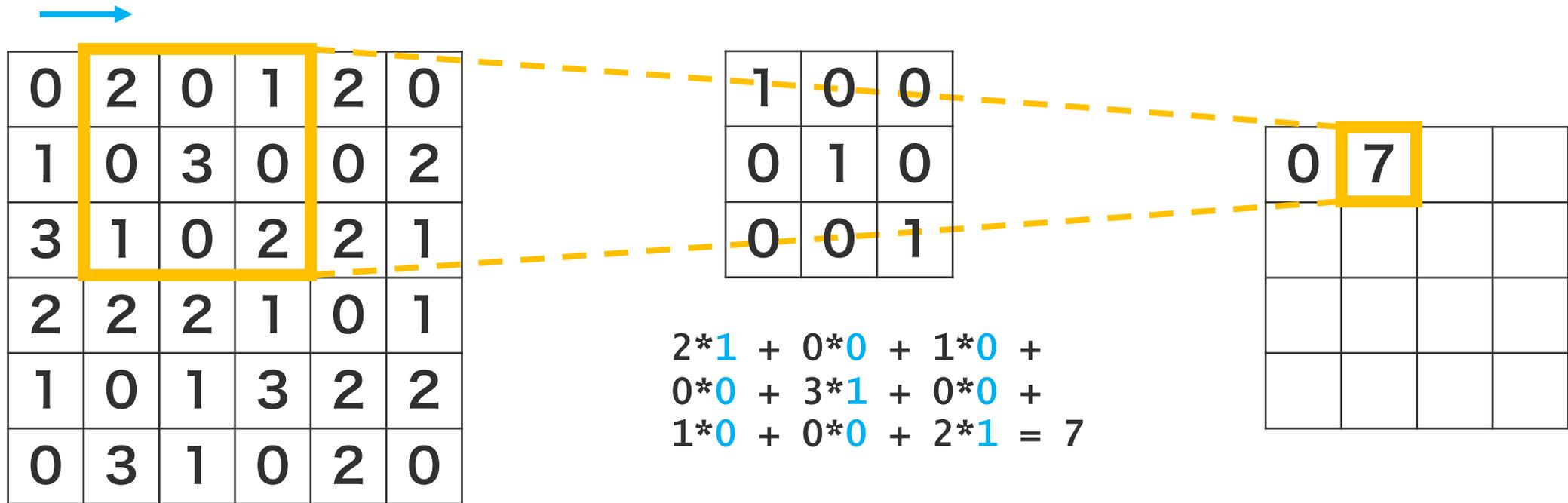
021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	037	093	052
031	051	053	081	041
093	049	058	092	026
054	101	061	024	101
109	023	026	105	022
111	042	024	102	102
151	149	123	101	153
251	254	255	245	253
254	253	250	249	251

3 channels

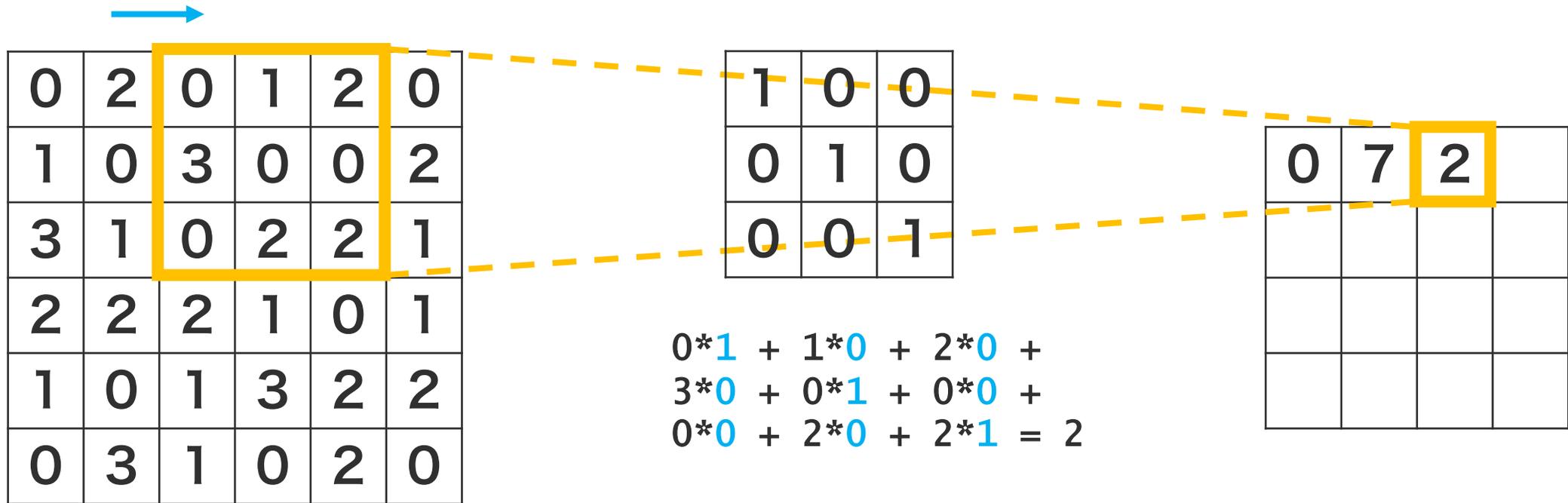
畳み込み演算



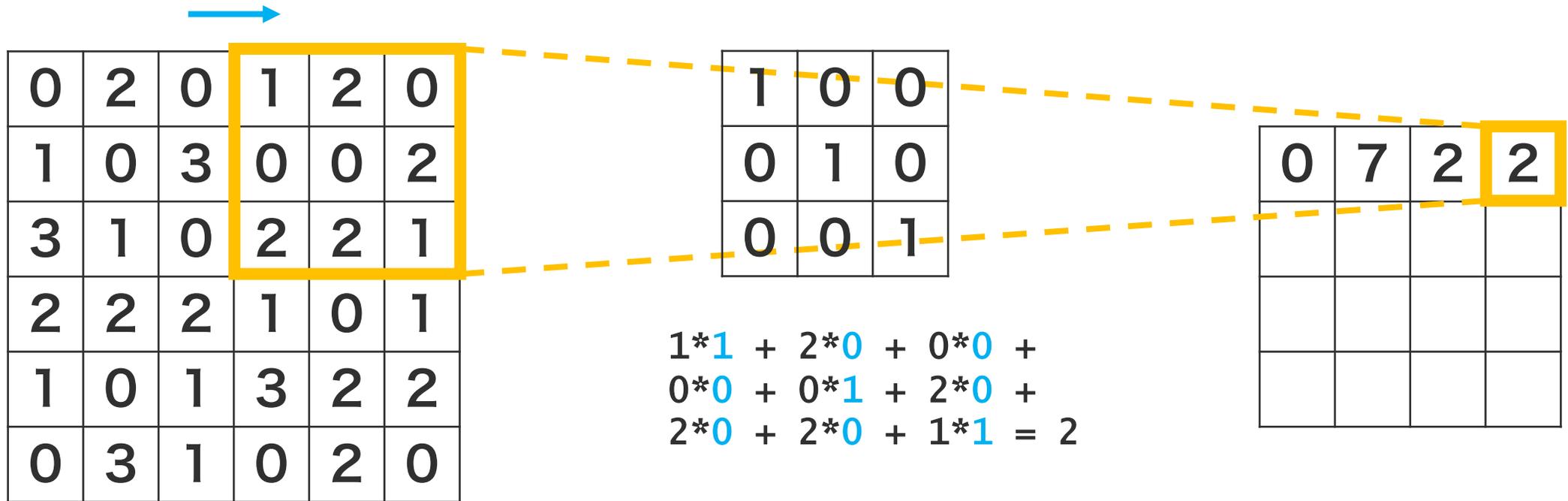
畳み込み演算



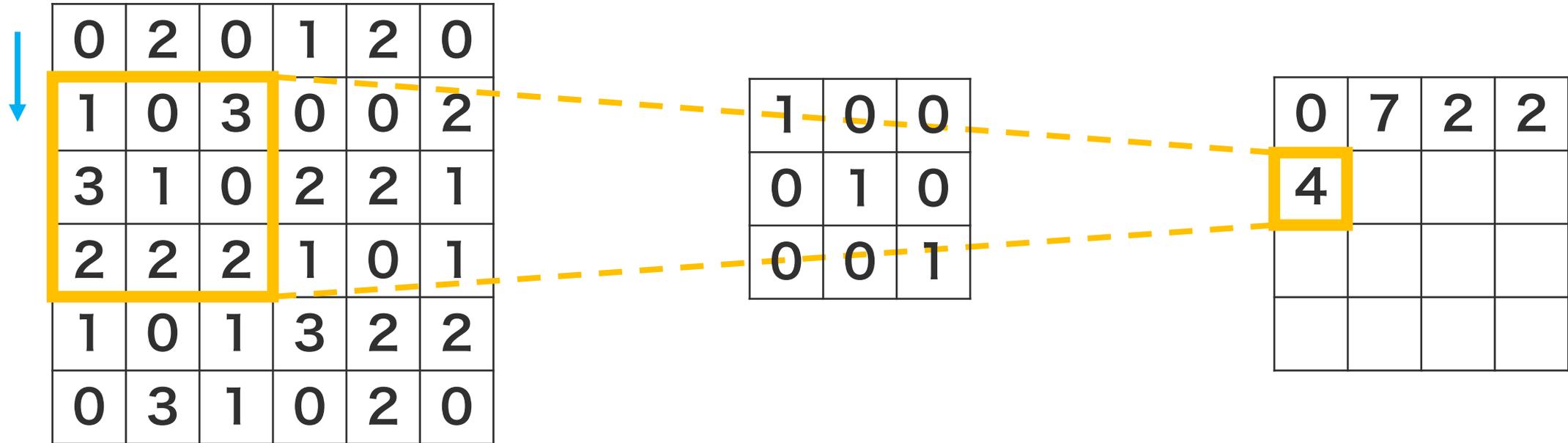
畳み込み演算



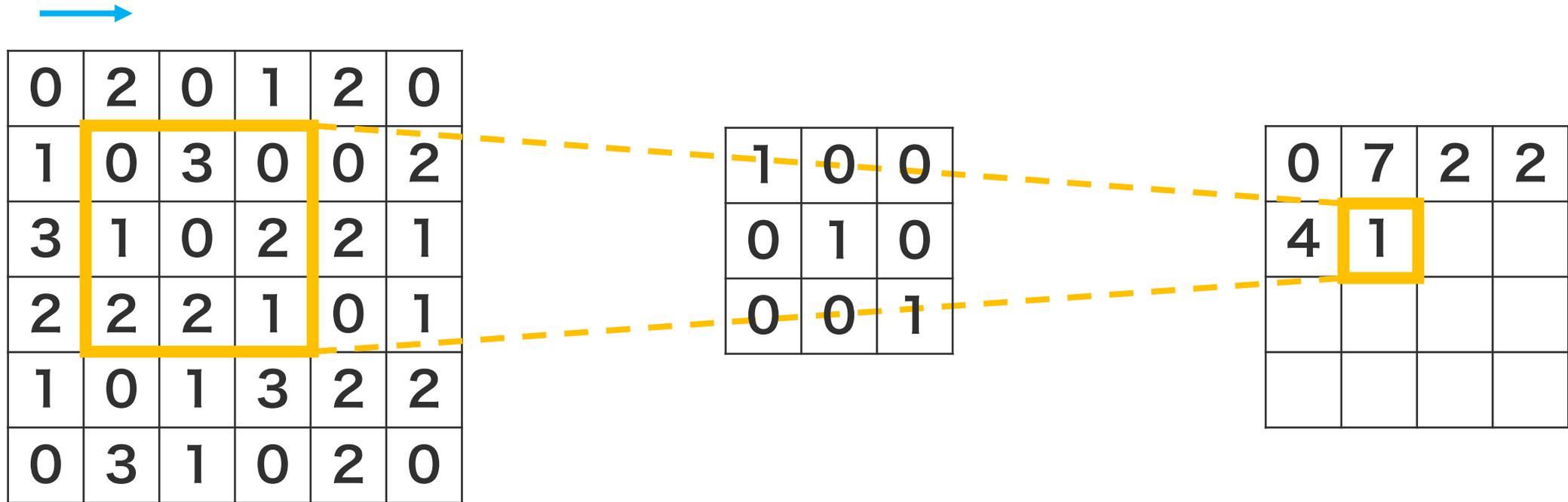
畳み込み演算



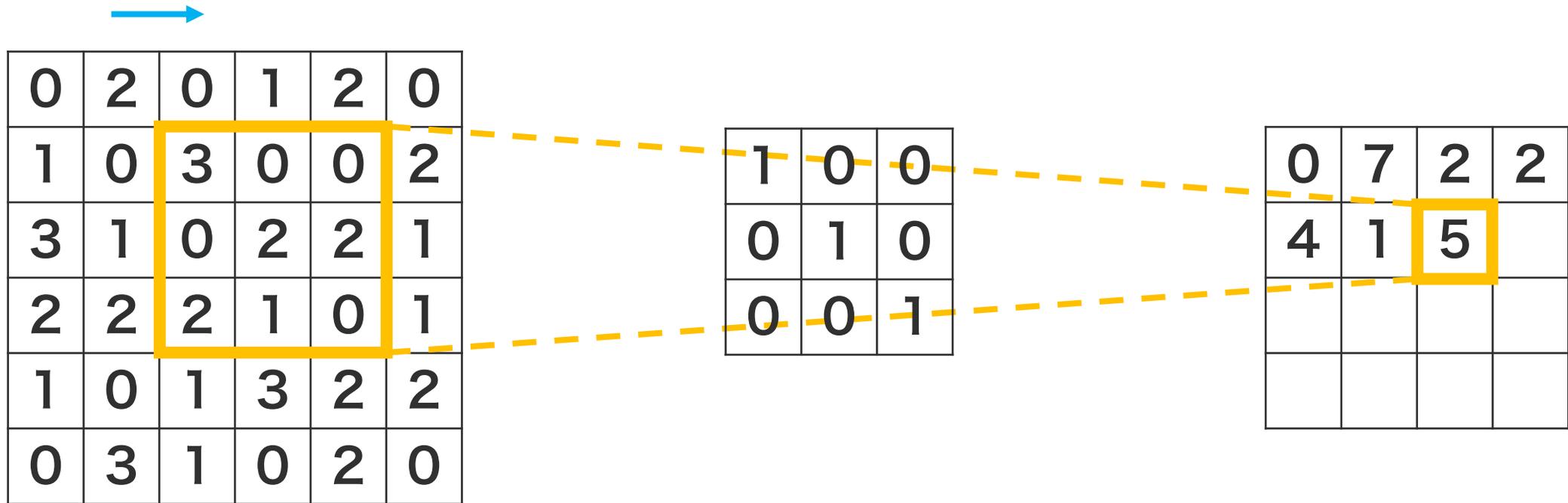
畳み込み演算



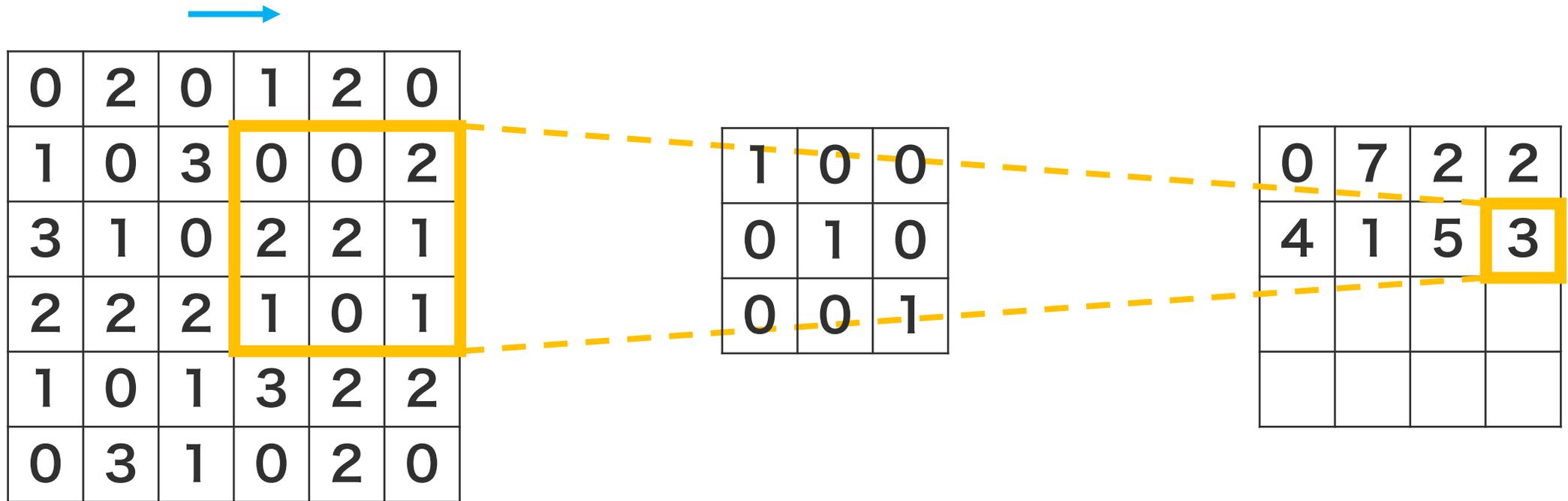
畳み込み演算



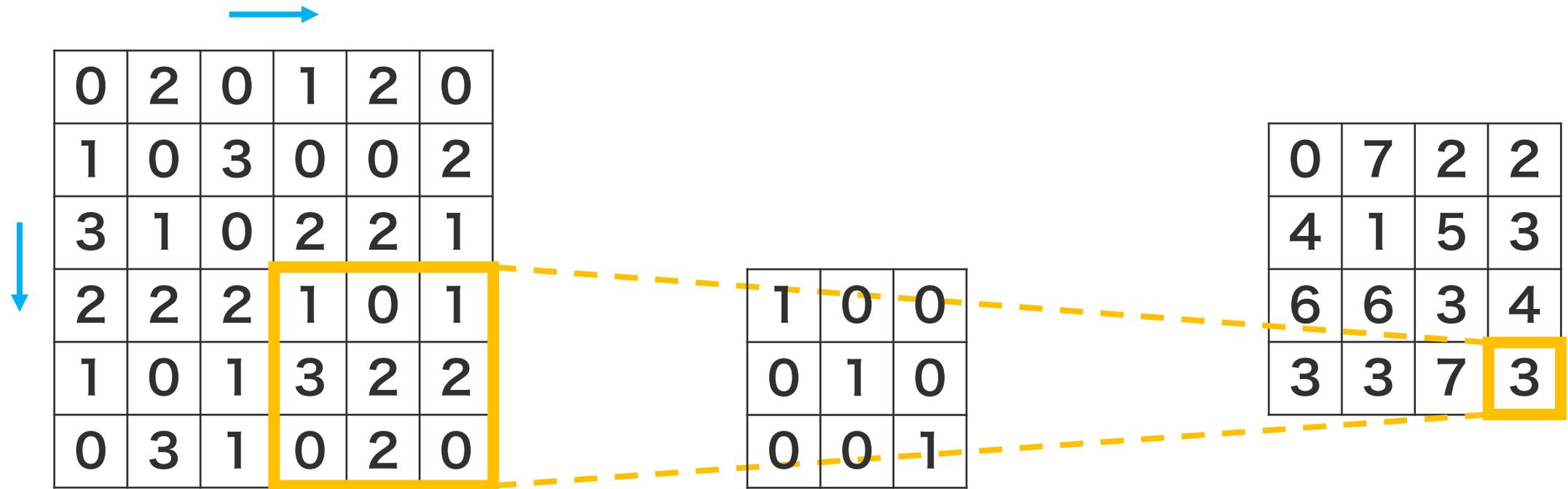
畳み込み演算



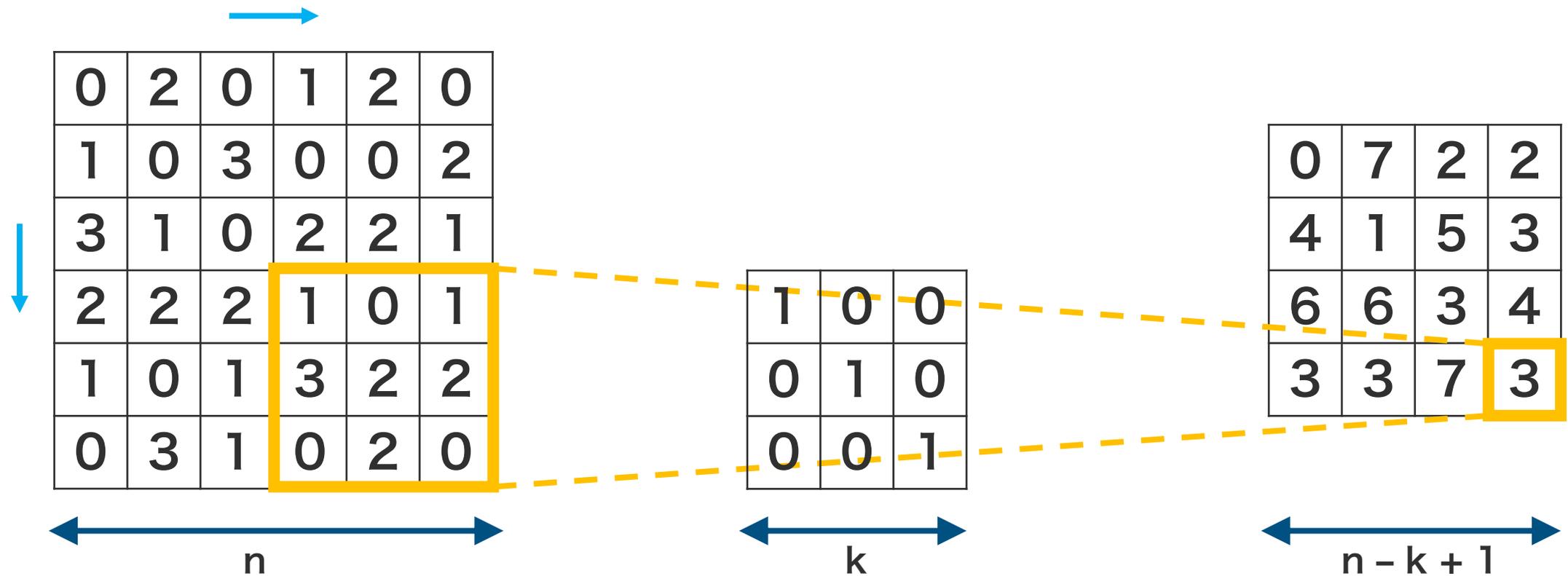
畳み込み演算



畳み込み演算



畳み込み演算



畳み込み演算を行うことによって、6x6 の入力行列が 4x4 の行列に縮小される。畳み込みを複数回行うことによって、画像サイズが大幅に小さくなる。また、画像中央にある特徴が極端に強調され、画像のエッジにある情報が徐々に消えてしまうリスクがある。

畳み込み演算

0	0	0	0	0	0	0	0
0	0	2	0	1	2	0	0
0	1	0	3	0	0	2	0
0	3	1	0	2	2	1	0
0	2	2	2	1	0	1	0
0	1	0	1	3	2	2	0
0	0	3	1	0	2	0	0
0	0	0	0	0	0	0	0

1	0	0
0	1	0
0	0	1

0	5	0	1	2	0
2	0	7	2	2	4
5	4	1	5	3	1
2	6	6	3	4	3
4	3	3	7	3	2
0	4	1	1	5	0

入力行列の周りにゼロを埋めることで、畳み込み後の行列のサイズは変化しない。

Grayscale Photo



```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

1 channel

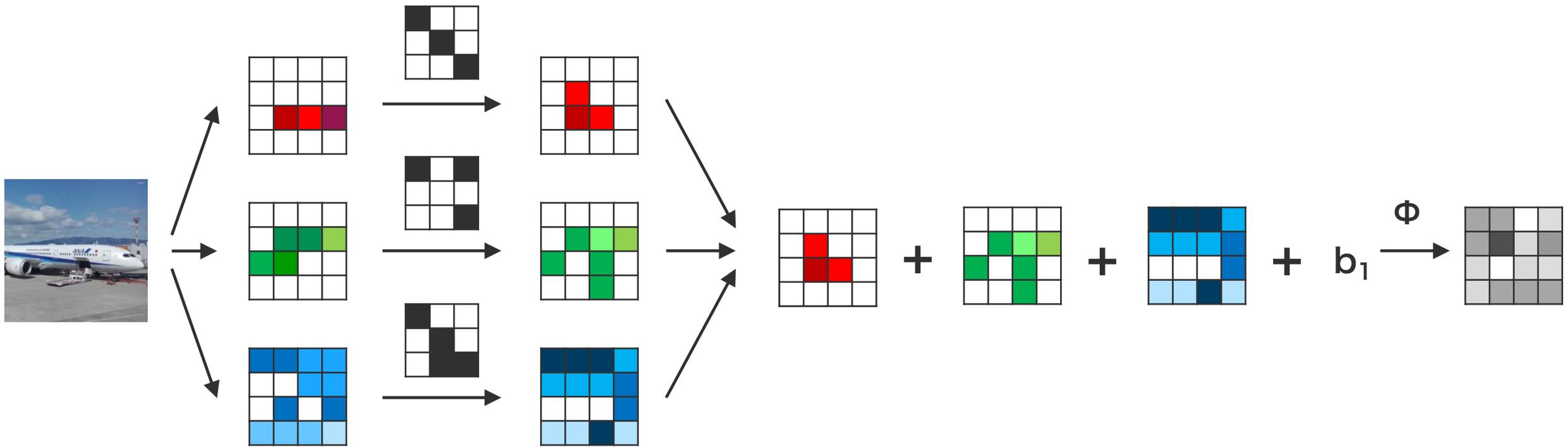
Color Photo



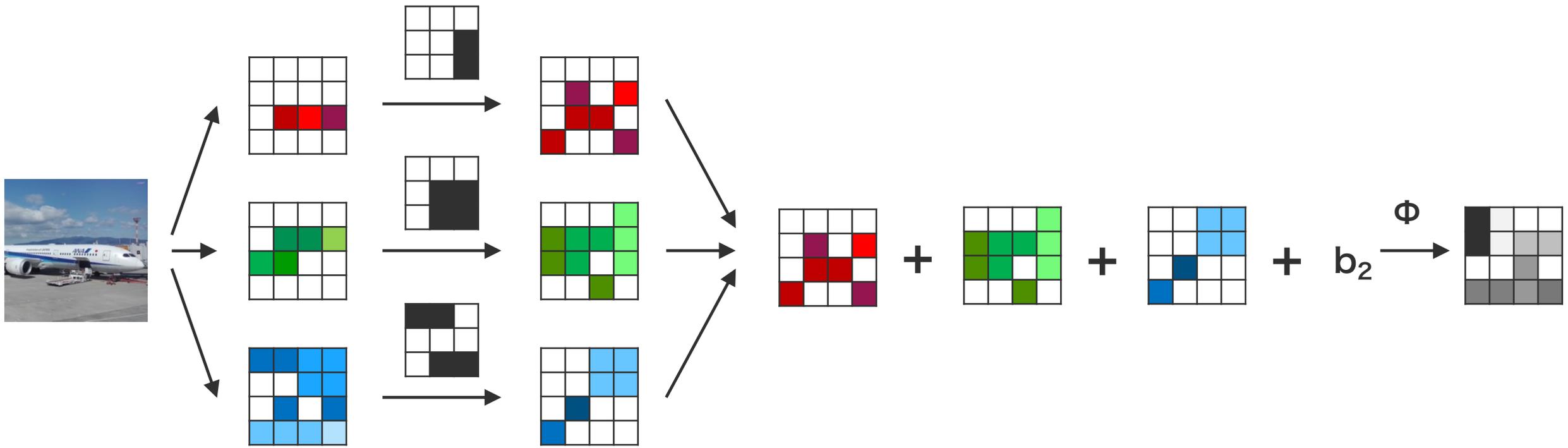
```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029 0224
025 061 012 091 03 052 1 243
031 051 053 081 041 0 217
093 049 058 092 026 0 236
054 101 061 024 102 1 216
109 023 026 052 022 2 220
111 042 024 021 024 7 231
151 251 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

3 channels

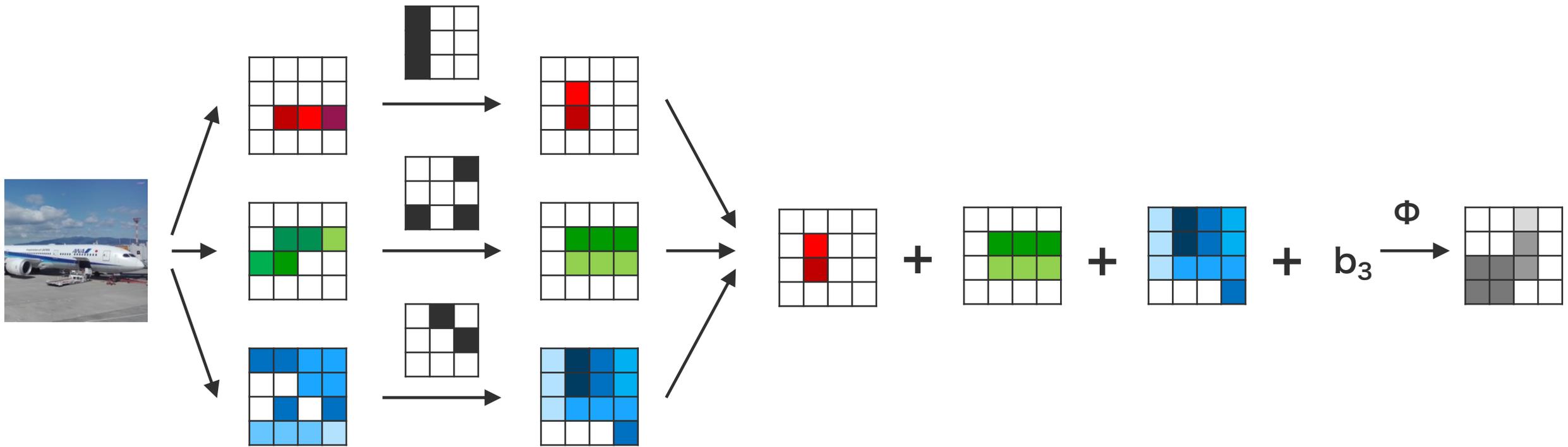
畳み込み演算



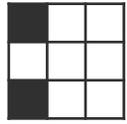
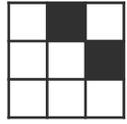
畳み込み演算



畳み込み演算

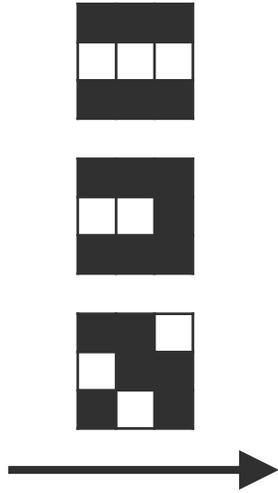


畳み込み演算



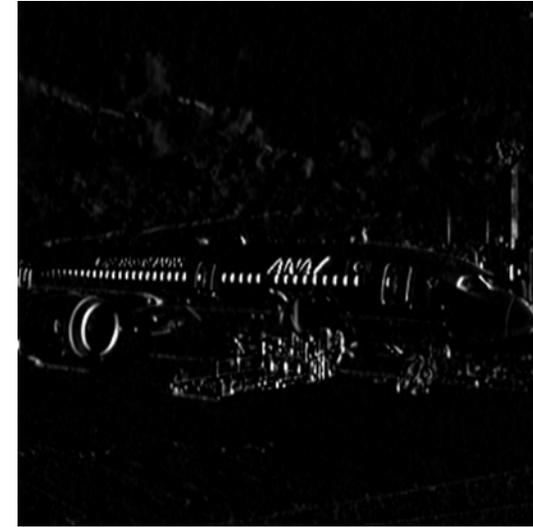
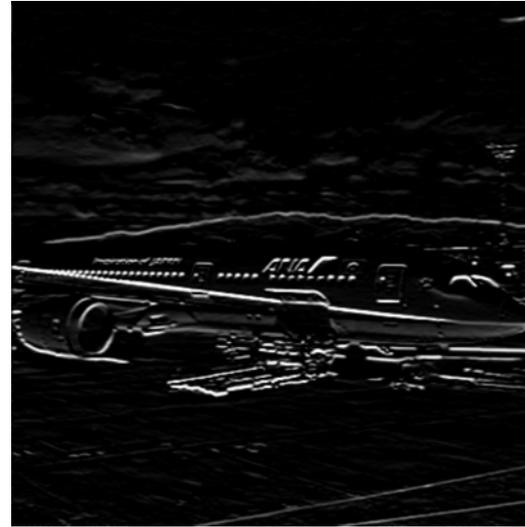
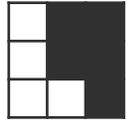
カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

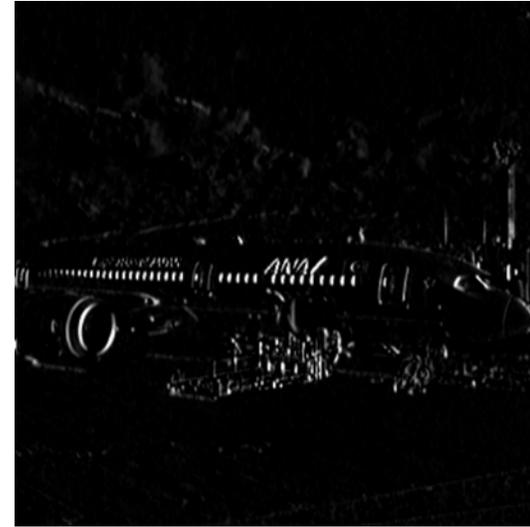
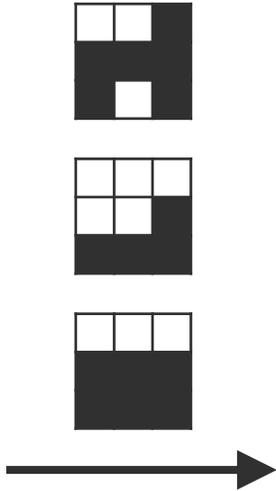
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。



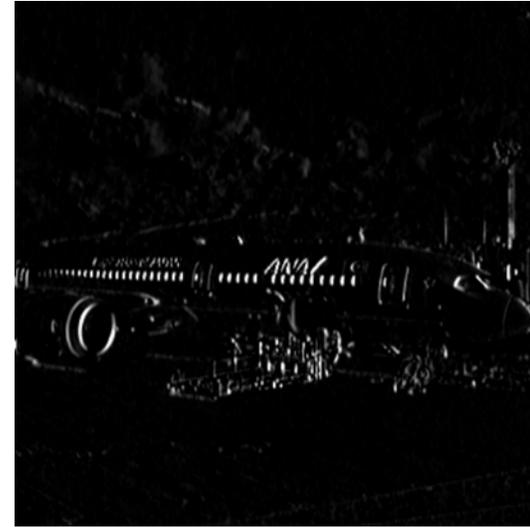
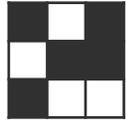
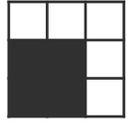
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。



畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

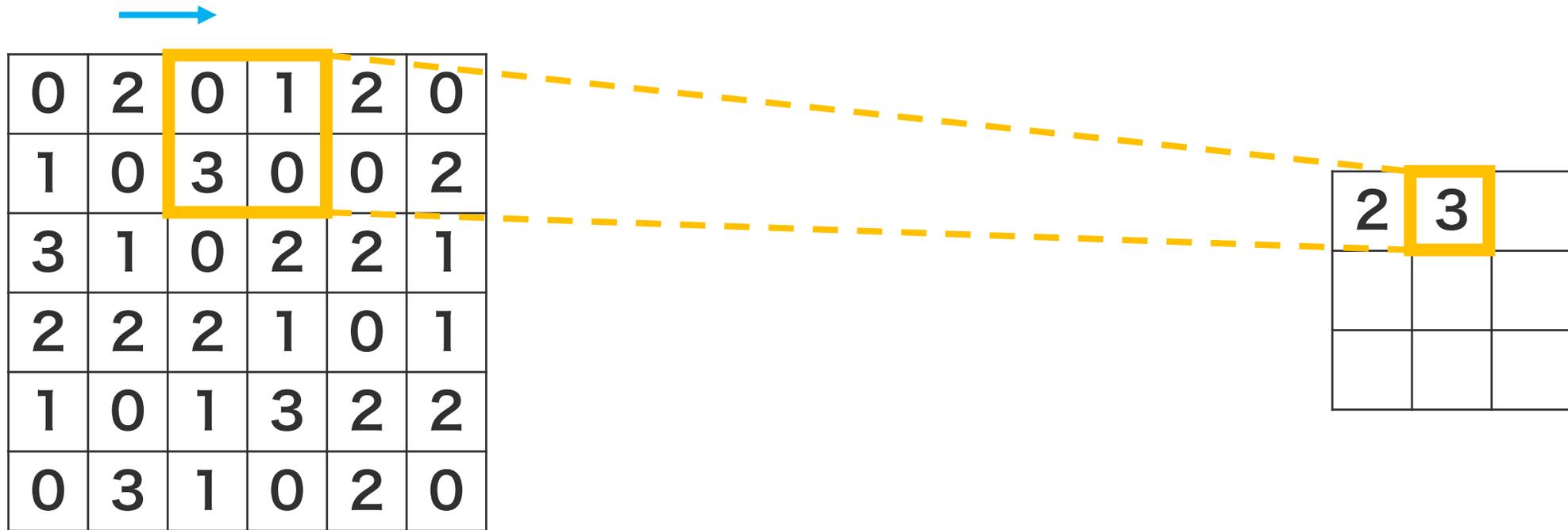


プーリング演算

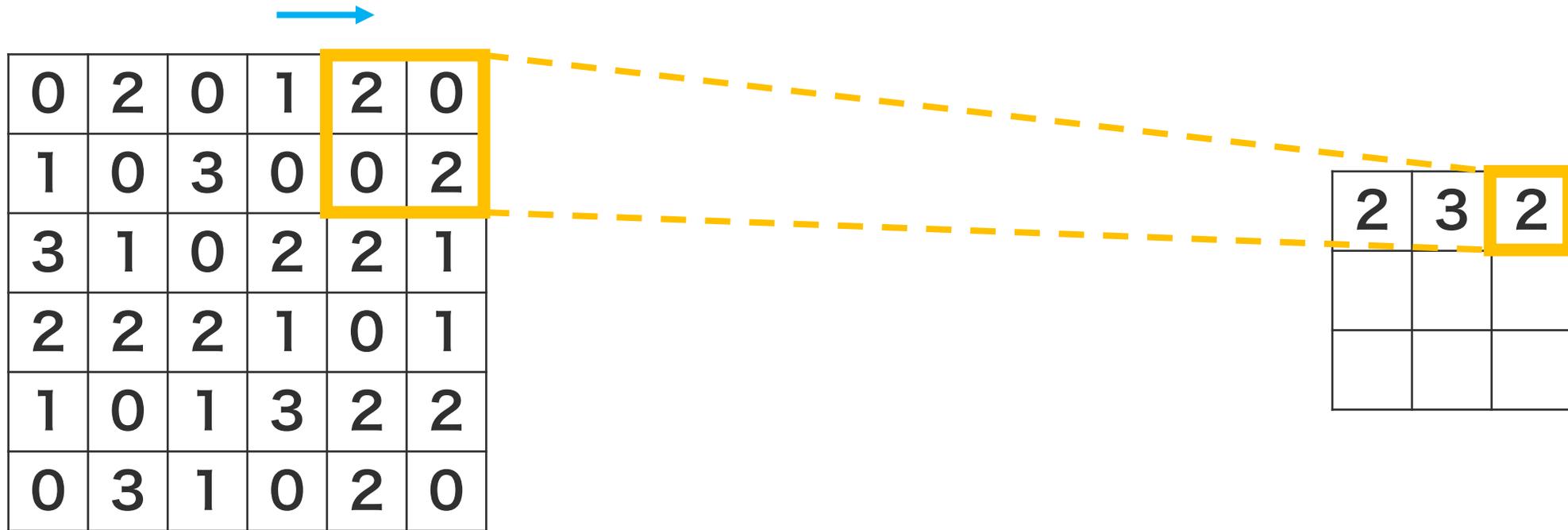
画像の局所的な部分が少し変化しても、その最大プーリング演算の結果が変化しないため、最大プーリングは入力データのノイズに対してより頑健な特徴量を生成するのに役立つ。



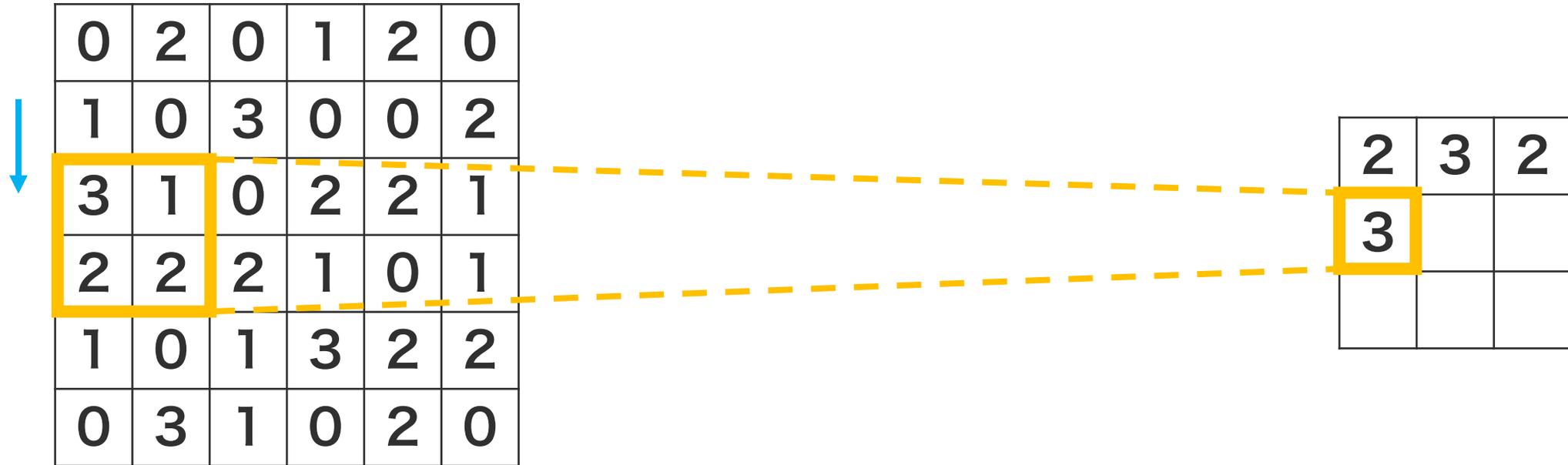
プーリング演算



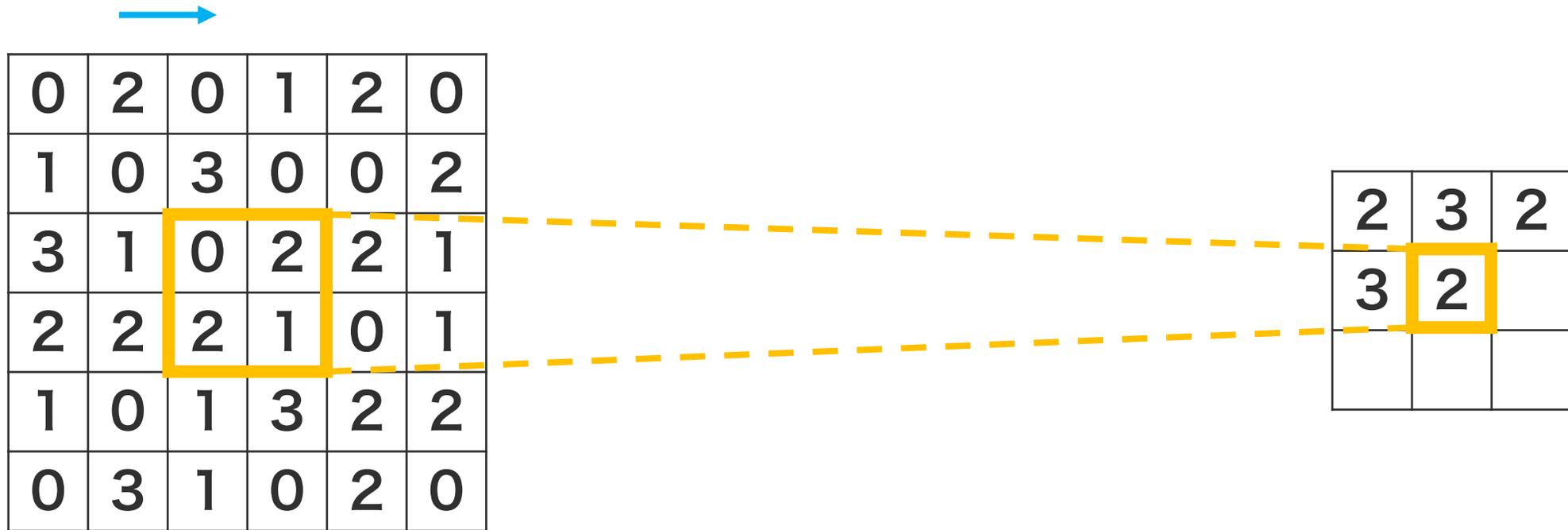
プーリング演算



プーリング演算



プーリング演算



プーリング演算



0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

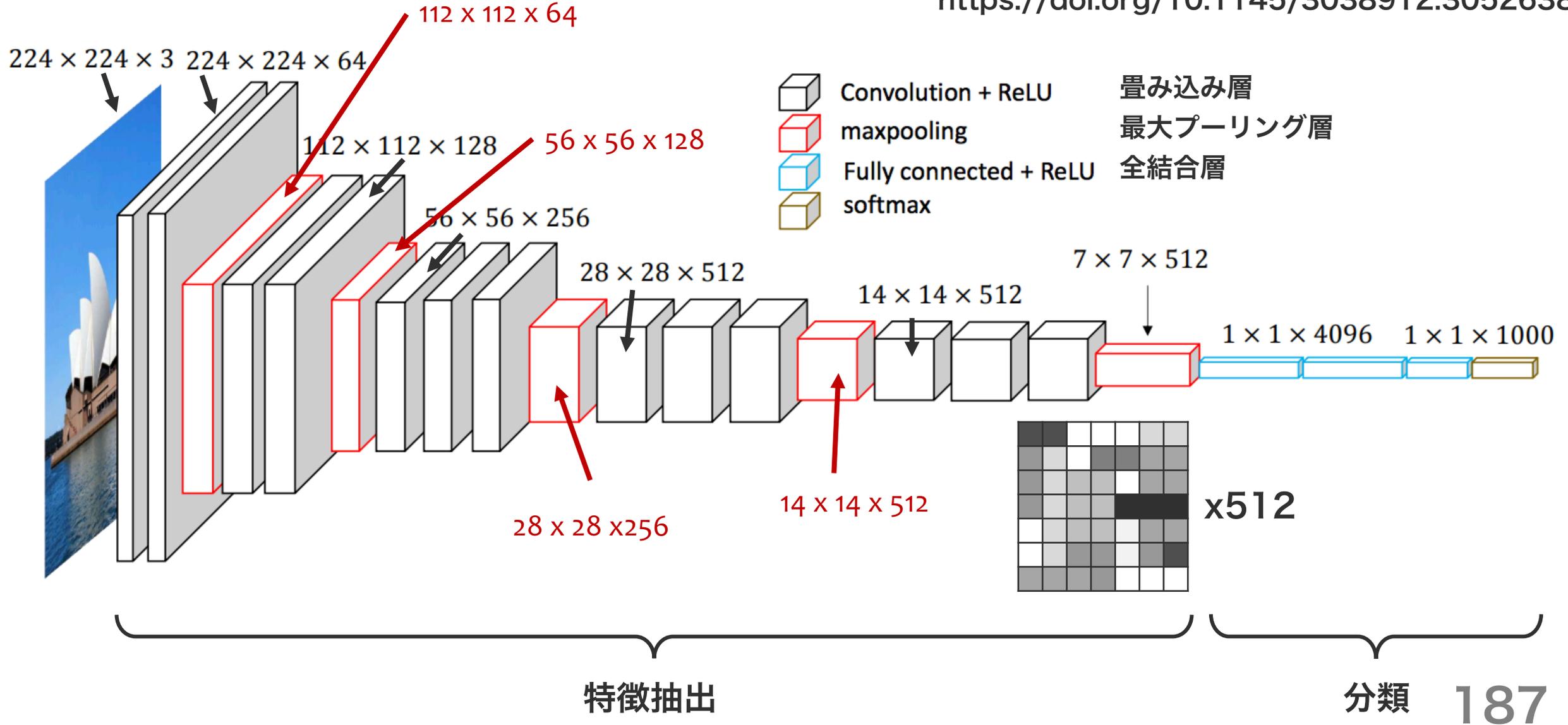
2	3	2
3	2	2

プーリング演算



VGG16

<https://doi.org/10.1145/3038912.3052638>



物体分類

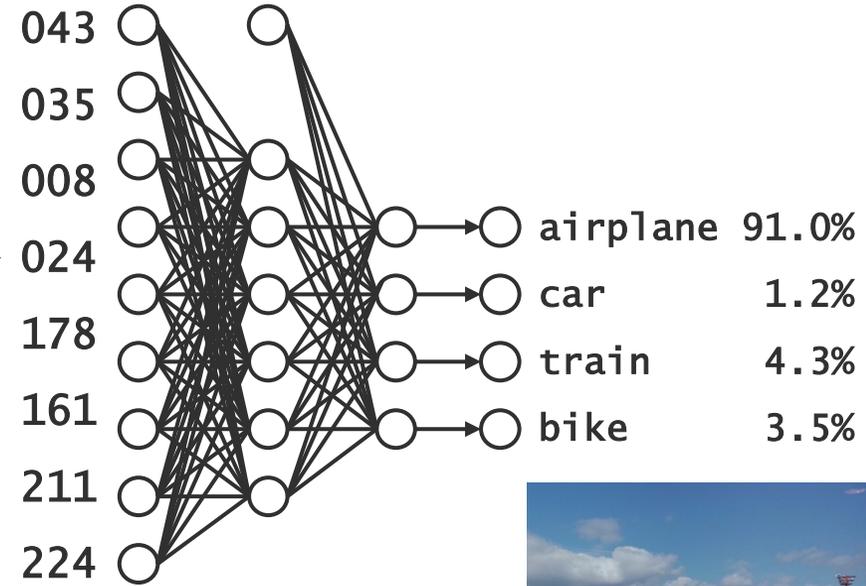
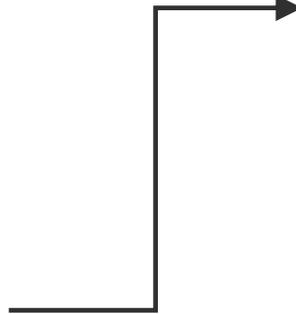
021 031 022 063 042
041606140314041303324
08268210256297202364324
04632452033493950324243
02579517913814029582617
03149512953398119417136
0934040058909122362016
05432015061523630412220
10949235031795109274331
11152425024594919344221
1525149254235502455253
254 253 250 249 251



021 031 022
244 250 251
04114951054034
2570145191650023
08220840946031
049182850241095
251832052102
01221694029124
251095098115
1940241056201
21022381031221
019 118 077
115 198 108



043 035
008 024
178 161
211 224



物体検出

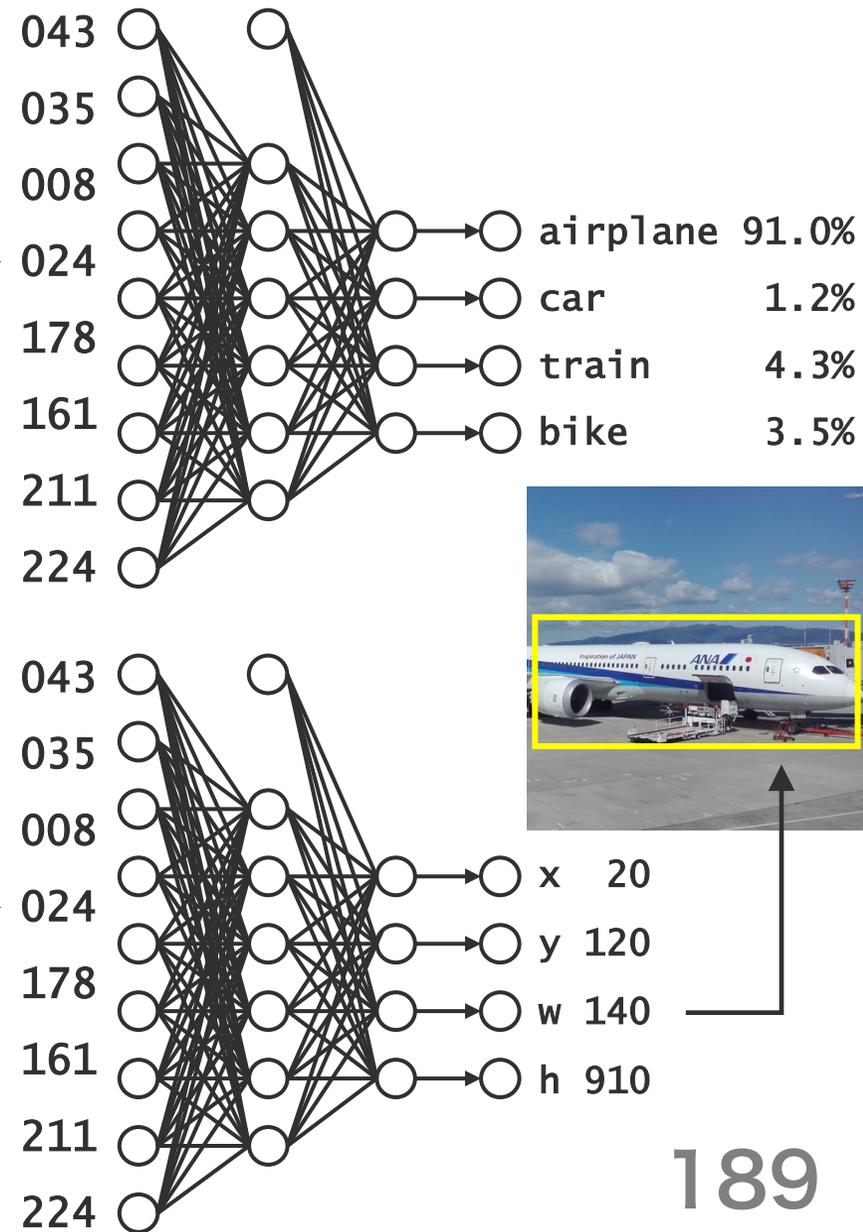
021 031 022 063 042
041606140314044303324
08268210256297202364324
04632452033493950324243
02579517913814029582617
03149512953398119417136
09340440058909122362016
05432015061503630412220
10949235030795109274331
11152535024502910344221
152514954235502455353
254 253 250 249 251



021 031 022
244 250 251
041110551054034
25214511650023
08220840976031
0491020810271095
251132082702
01221619029124
251109098115
19402104056201
21022381031221
019 118 077
115 198 108

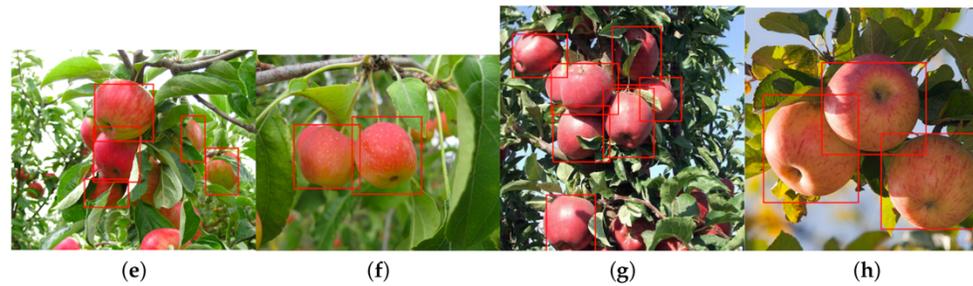
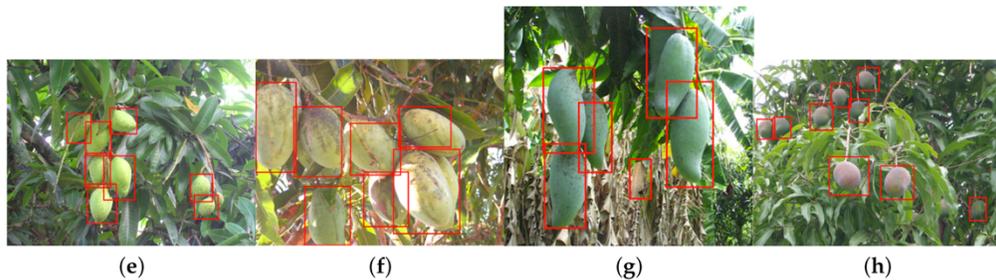
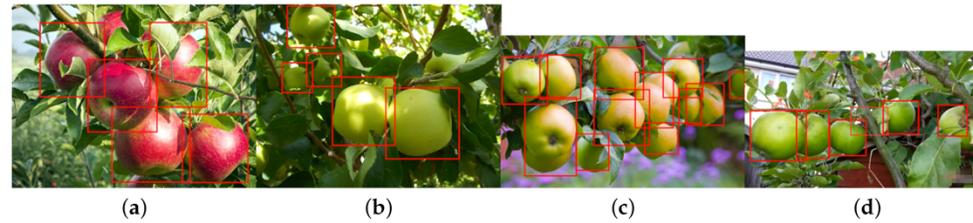
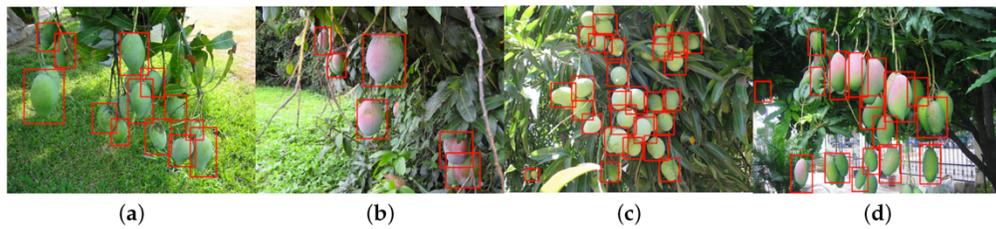
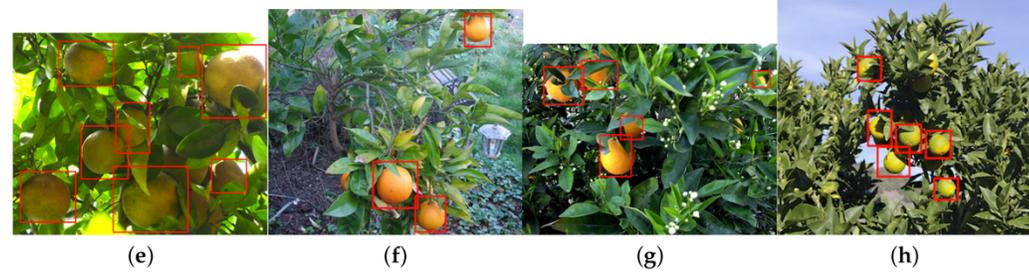
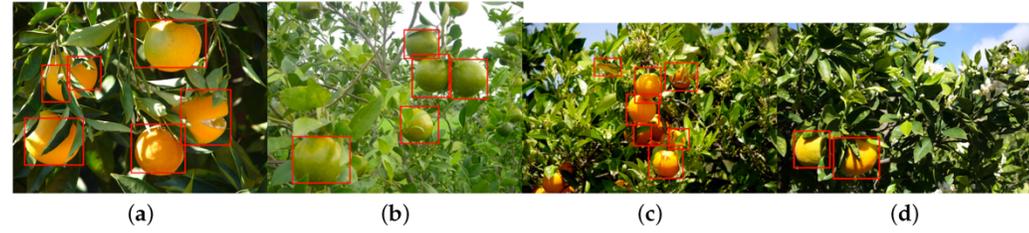
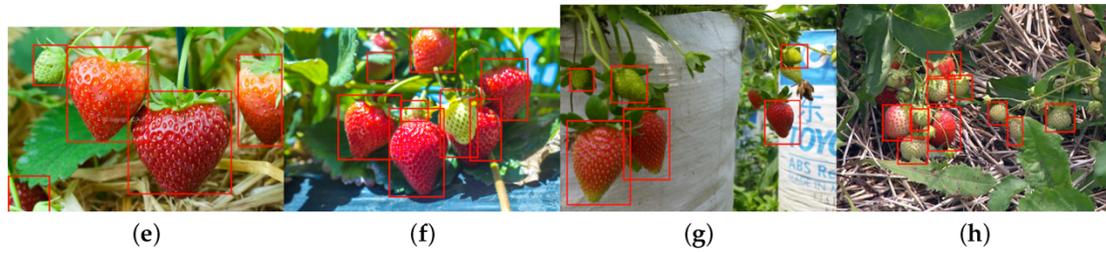
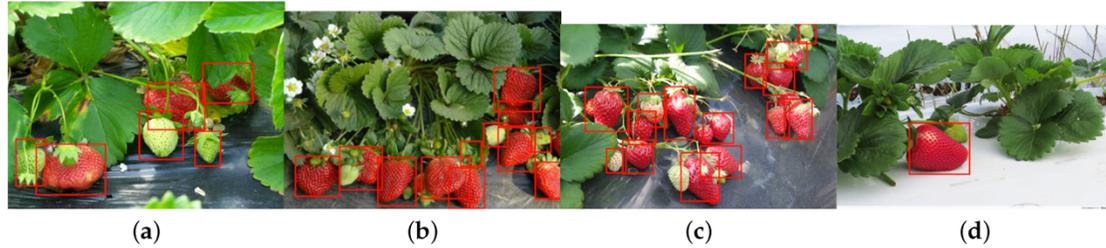


043 035
008 024
178 161
211 224



研究事例 / fruits detection

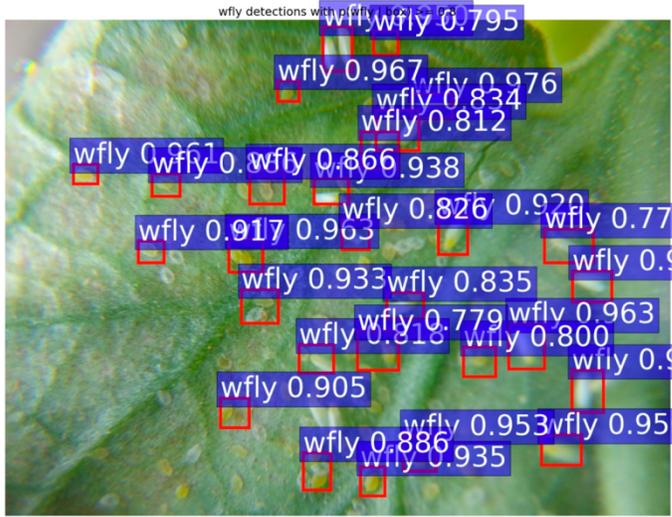
DeepFruits: A Fruit Detection System Using Deep Neural Networks. Sensors 2016, 16(8):1222
<https://doi.org/10.3390/s16081222>



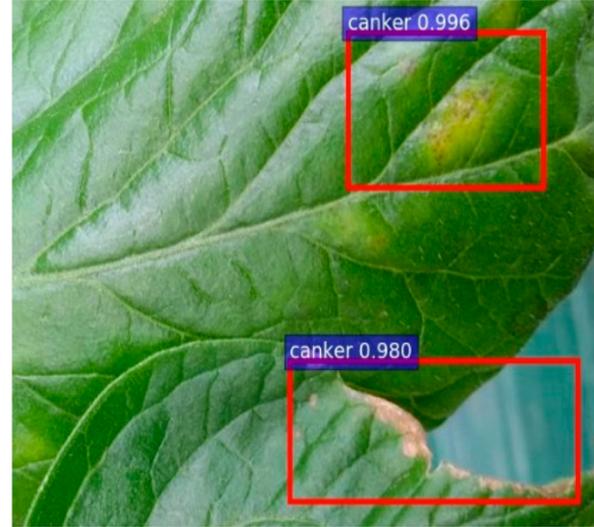
研究事例 / diseases detection

A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. Sensors 2017, 17(9):2022

<https://doi.org/10.3390/s17092022>



(a)



(b)



(e)



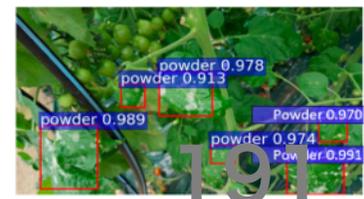
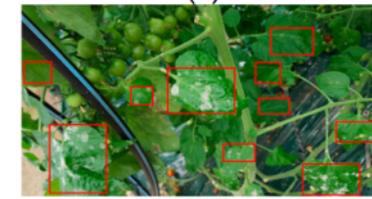
(f)



(g)

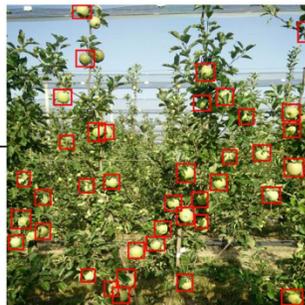
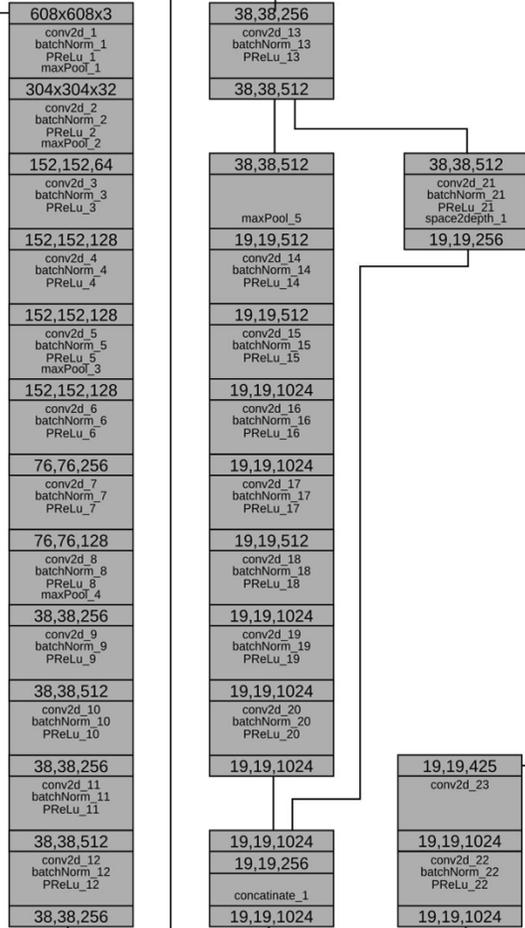


(h)



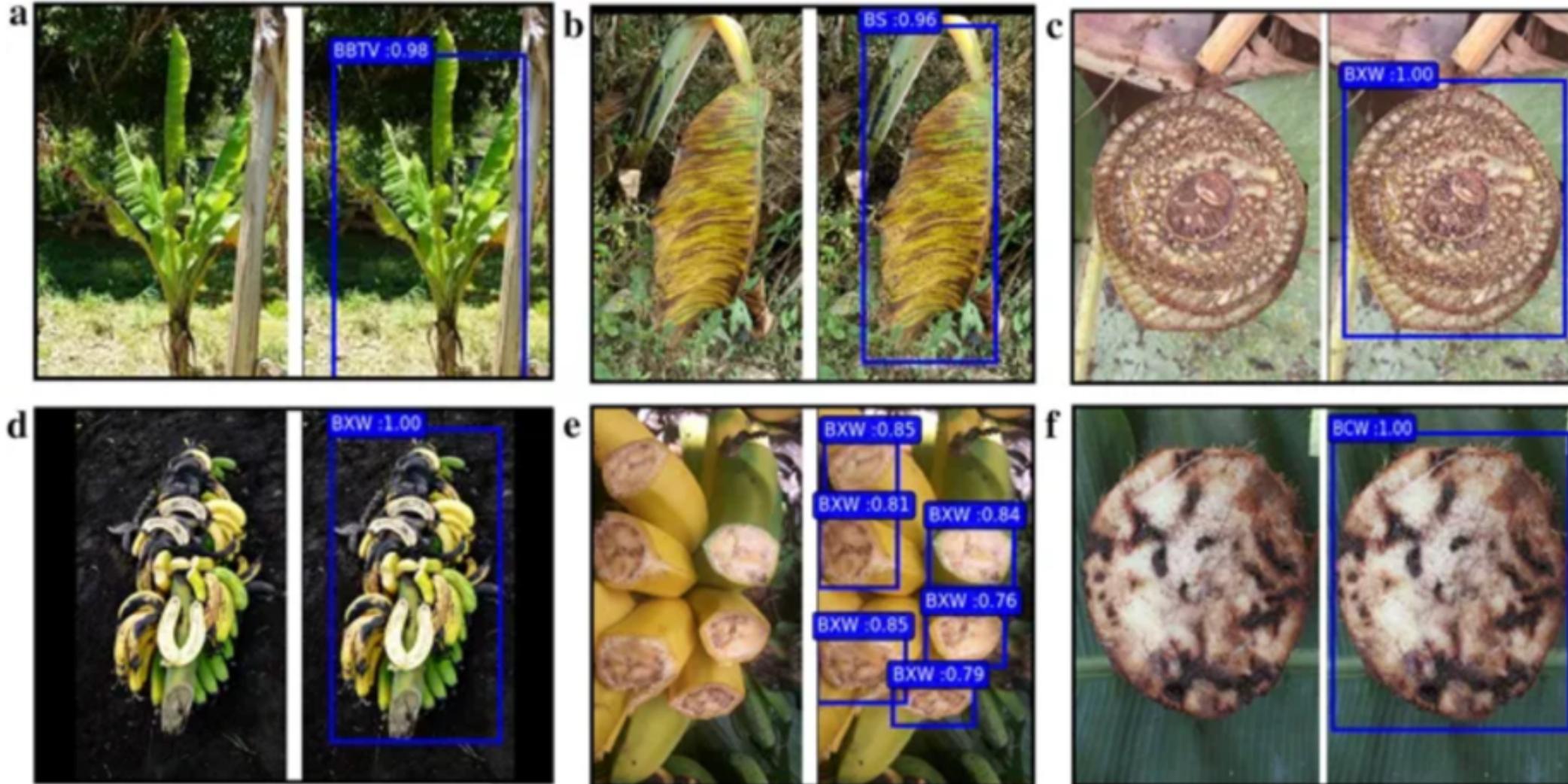
研究事例 / fruits detection

Single-Shot Convolution Neural Networks for Real-Time Fruit Detection Within the Tree. Front Plant Sci. 2019, 10:611
<https://doi.org/10.3389/fpls.2019.00611>



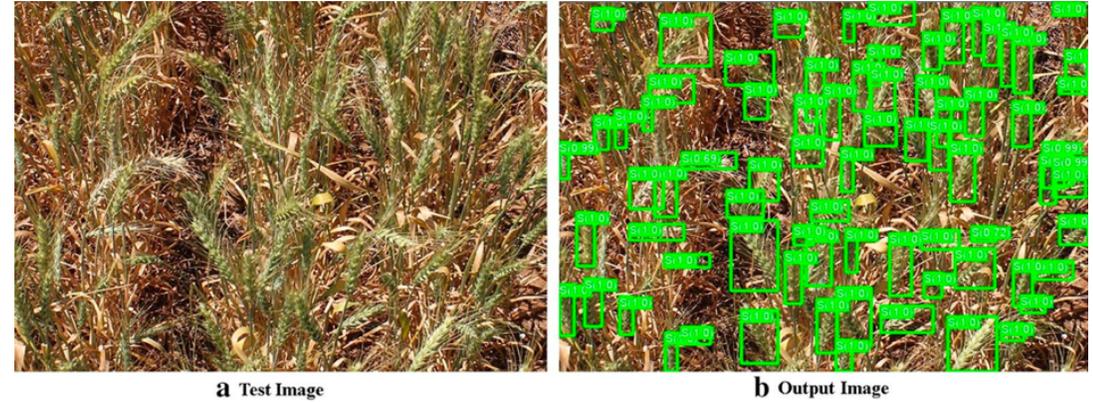
研究事例 / diseases detection

AI-powered banana diseases and pest detection. Plant Methods 2019, 15:92
<https://doi.org/10.1186/s13007-019-0475-z>



研究事例 / spikes detection

Detection and analysis of wheat spikes using Convolutional Neural Networks. Plant Methods 2018, 14:100
<https://doi.org/10.1186/s13007-018-0366-8>



negative →

positive →

testImages →

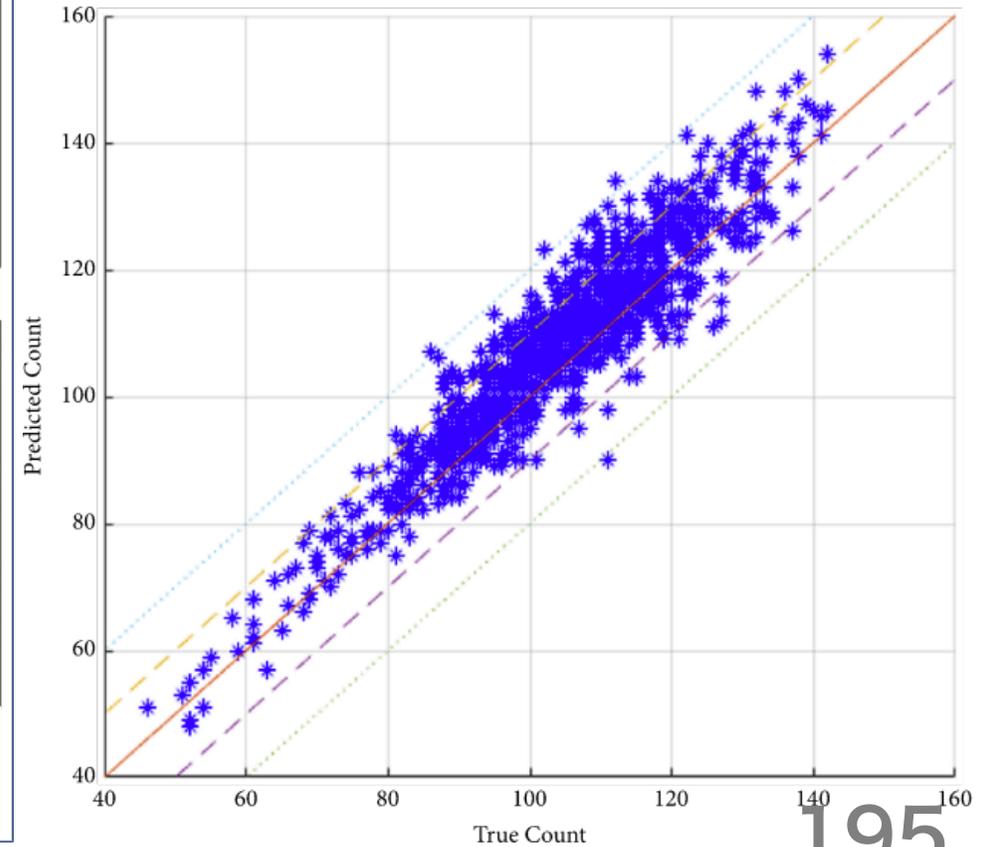
roi annotations	labels	bounding boxes			
class_map.txt	S	565	404	673	487
test_img_file.txt	S	703	250	807	371
test_roi_file.txt	S	915	12	1016	129
train_img_file.txt	S	653	654	824	888
train_roi_file.txt	S	835	746	902	858

Test images	Base	Extended model		
	GSYC	+ GSGC	+ YSYC	GSYC++
GSYC (20)	0.6653	0.6462	0.6435	0.6575
GSGC (5)	0.6570	0.7077	0.6405	0.6857
YSYC (5)	0.6546	0.6590	0.7163	0.7085
GSYC + GSGC + YSYC (30)	0.6050	0.6413	0.6520	0.6763

研究事例 / spikes detection

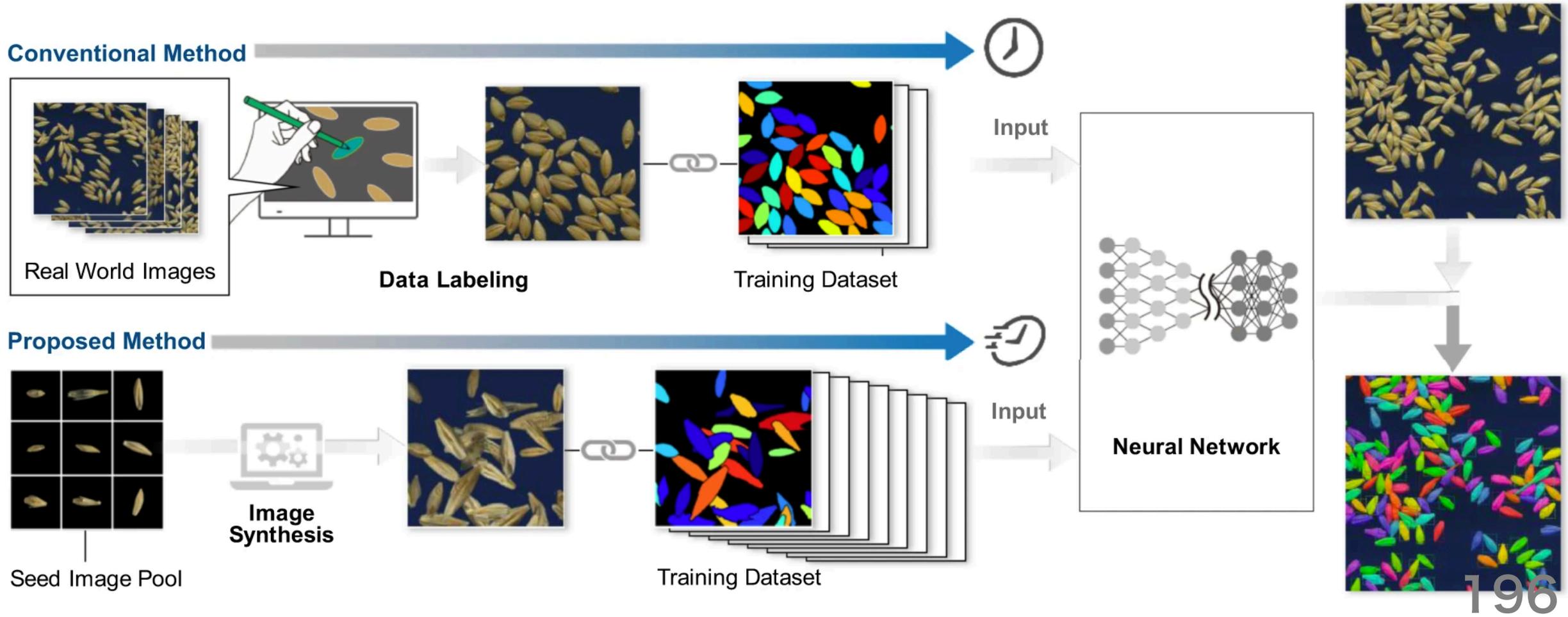
A Weakly Supervised Deep Learning Framework for Sorghum Head Detection and Counting. Plant Phenomics 2019, 1525874

<https://doi.org/10.34133/2019/1525874>



研究事例 / augmentation

Training instance segmentation neural network with synthetic datasets for crop seed phenotyping. Communications Biology 2020, 3:173
<https://doi.org/10.1038/s42003-020-0905-5>



研究事例 / disease detection

Convolutional Neural Networks for the Automatic Identification of Plant Diseases. *Frontiers in Plant Science* 2019, 10:941
<https://doi.org/10.3389/fpls.2019.00941>

