

# 肺のレントゲン画像を用いた深層学習

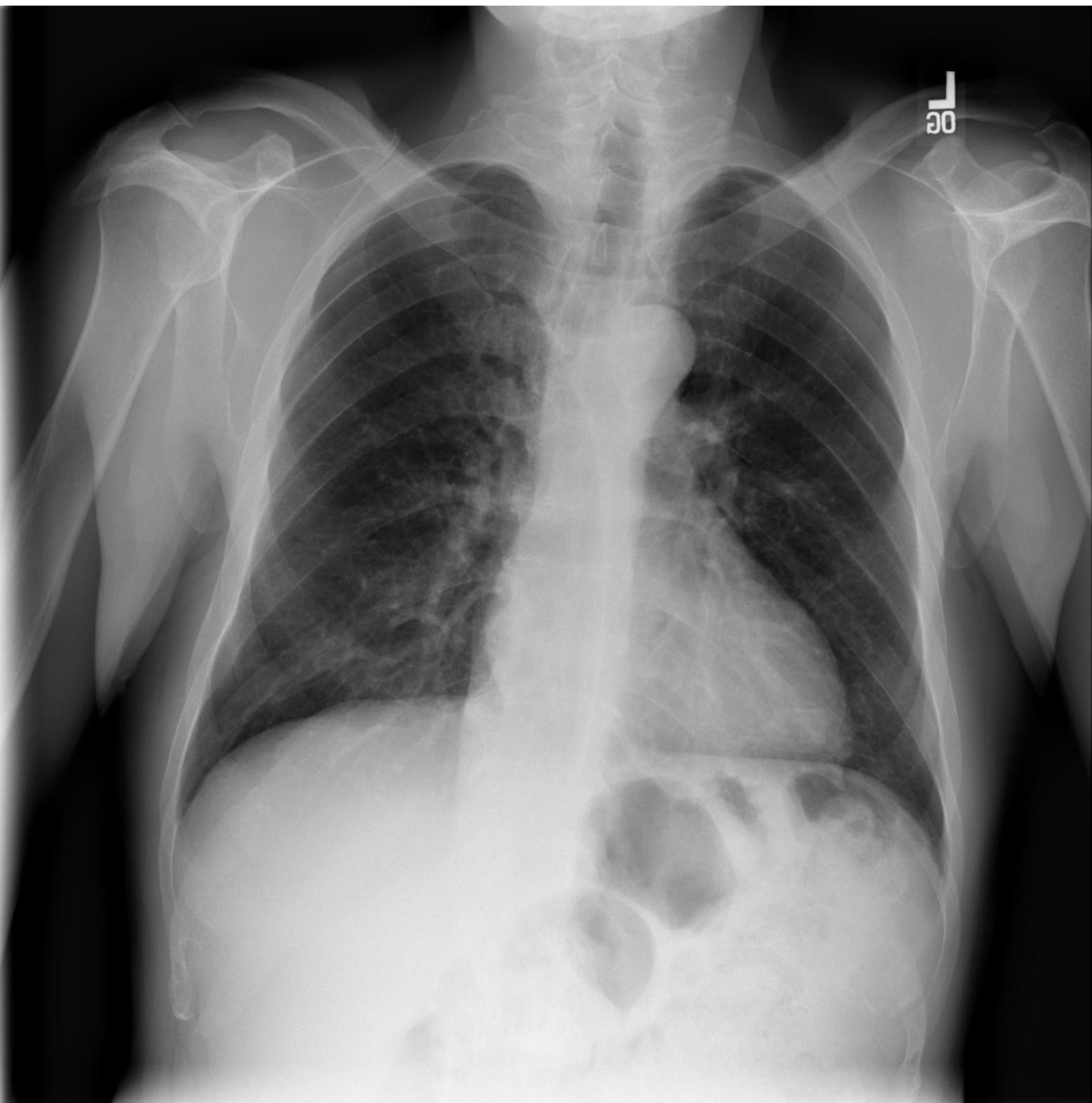
本教材を使用した際にはお手数ですが、  
下記アンケートフォームにご協力下さい。



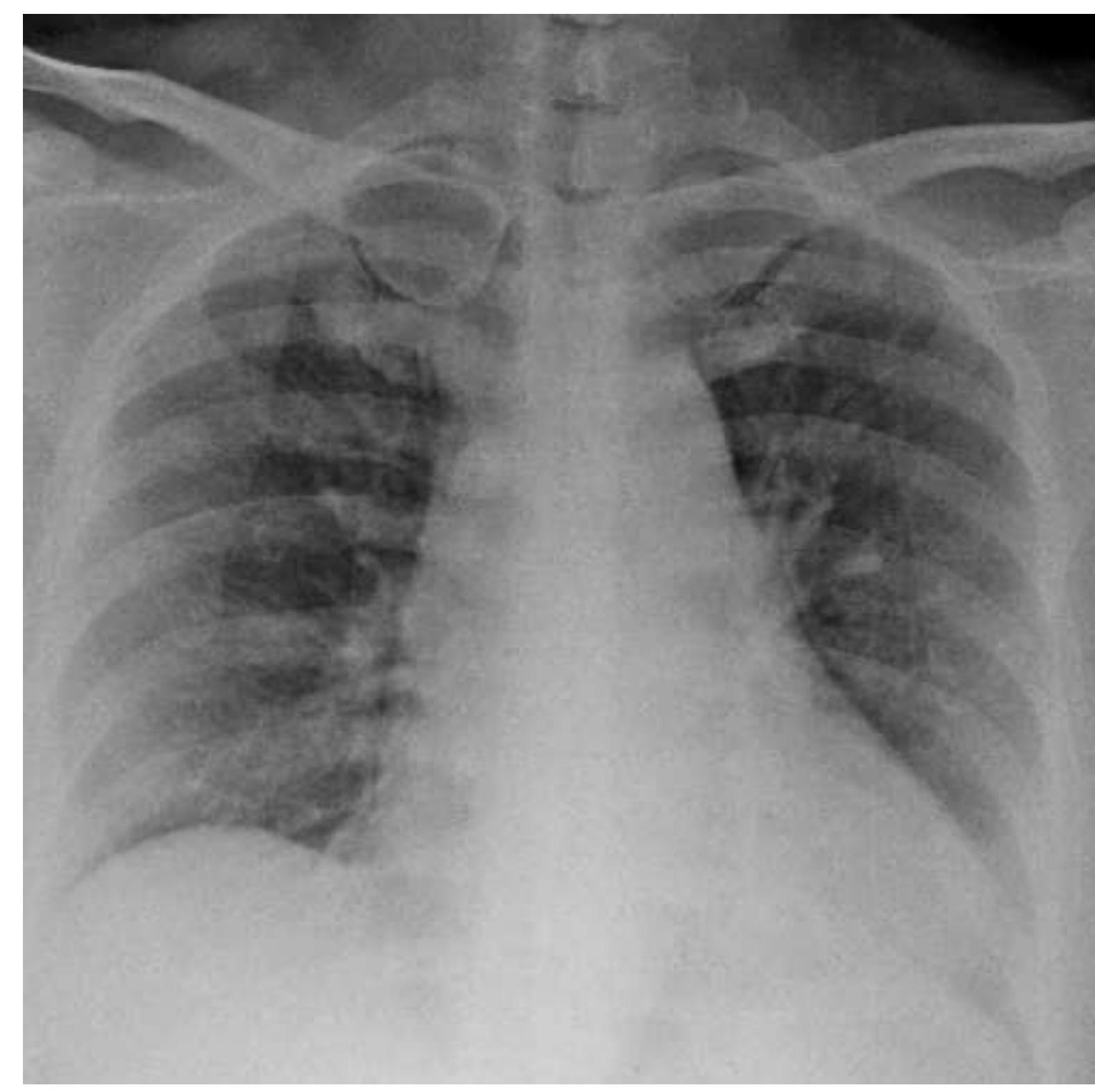
統合教育機構  
須藤毅顕

# 肺のレントゲン画像で深層学習

健康



肺炎

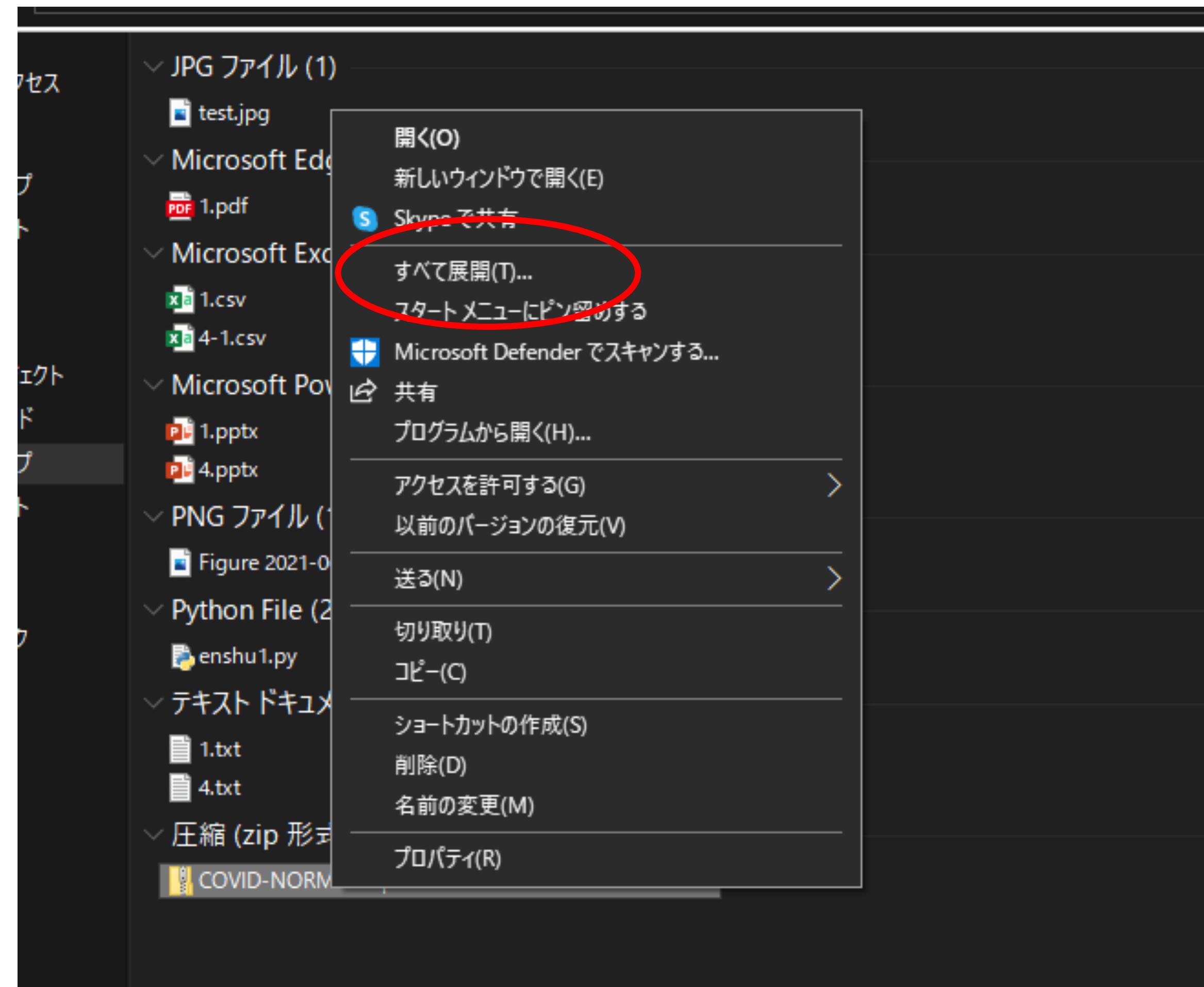


# 肺のレントゲン画像で深層学習



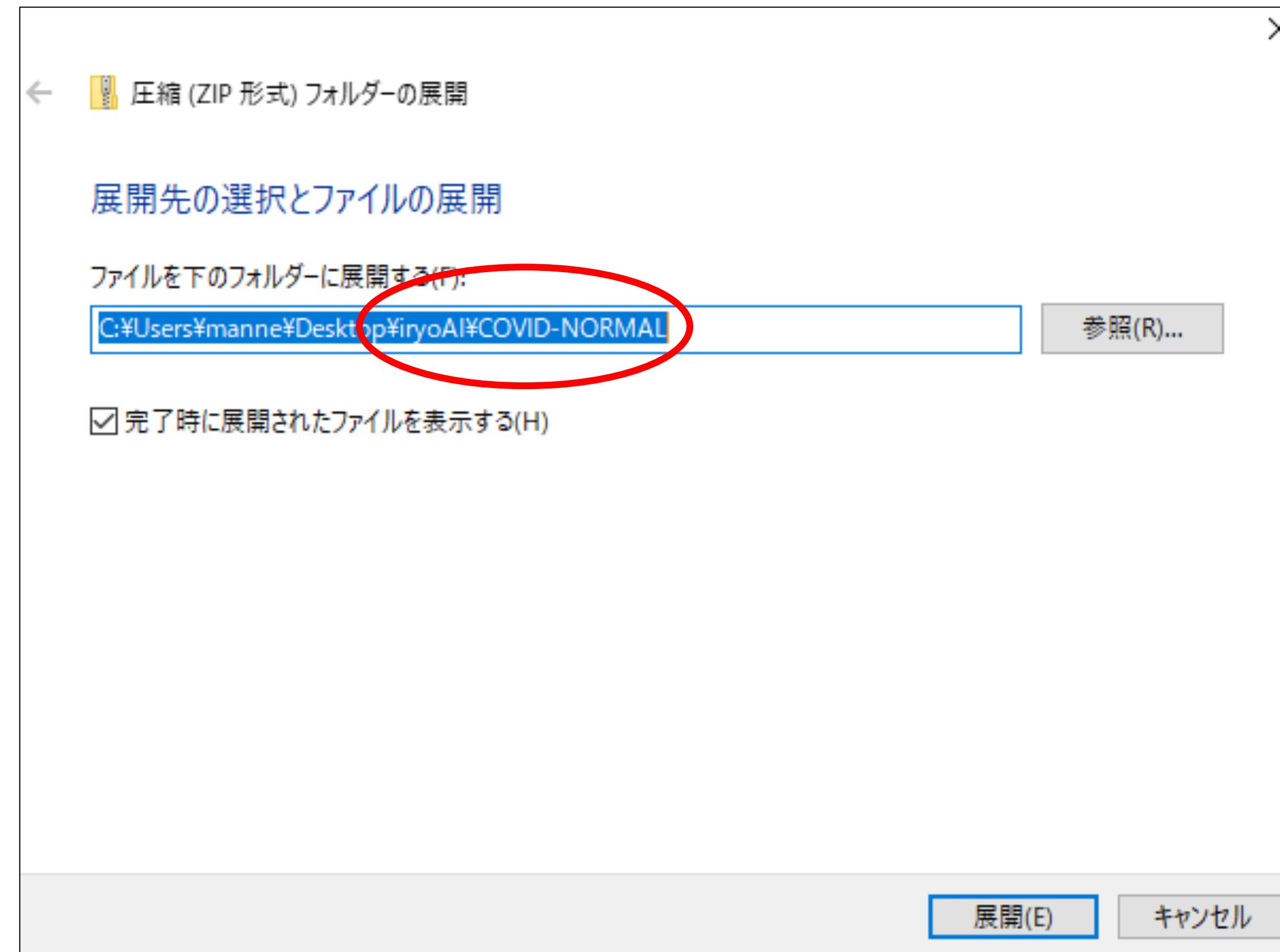
# 肺のレントゲン画像で深層学習

先にCOVID-NORMAL.zipのファイルを解凍しましょう



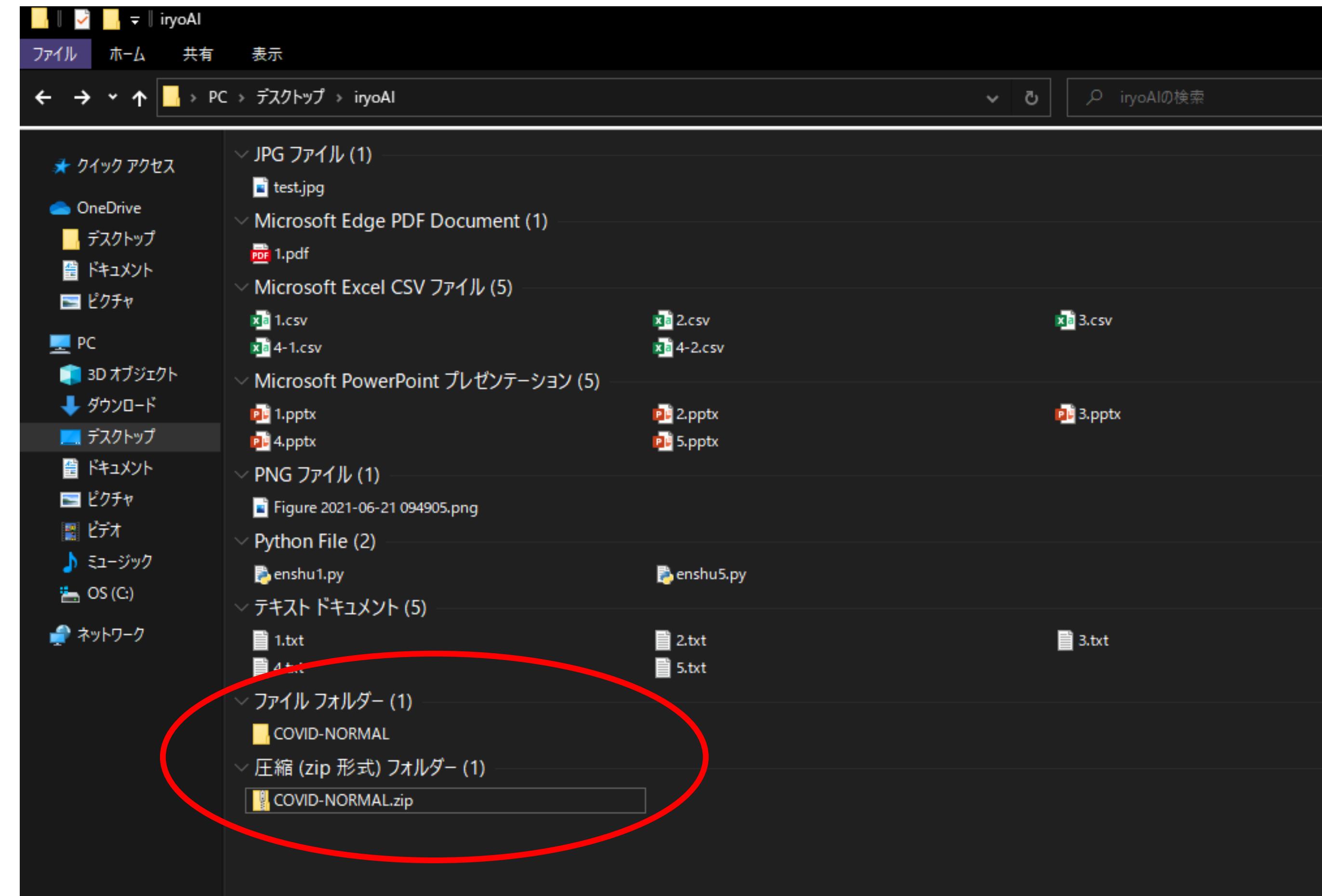
# 肺のレントゲン画像で深層学習

“¥iryoAI¥COVID-NORMAL”になっていることを確認

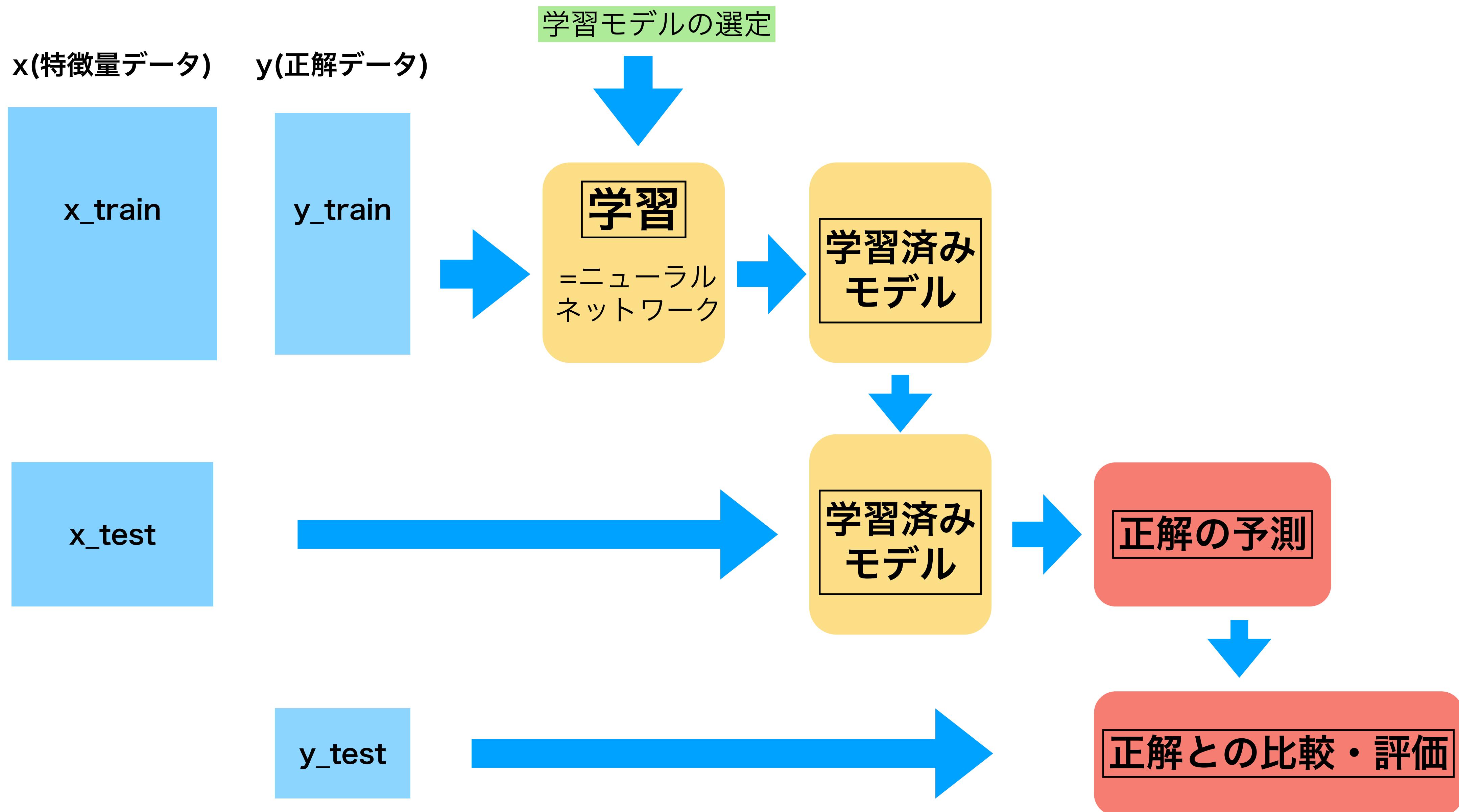


# 肺のレントゲン画像で深層学習

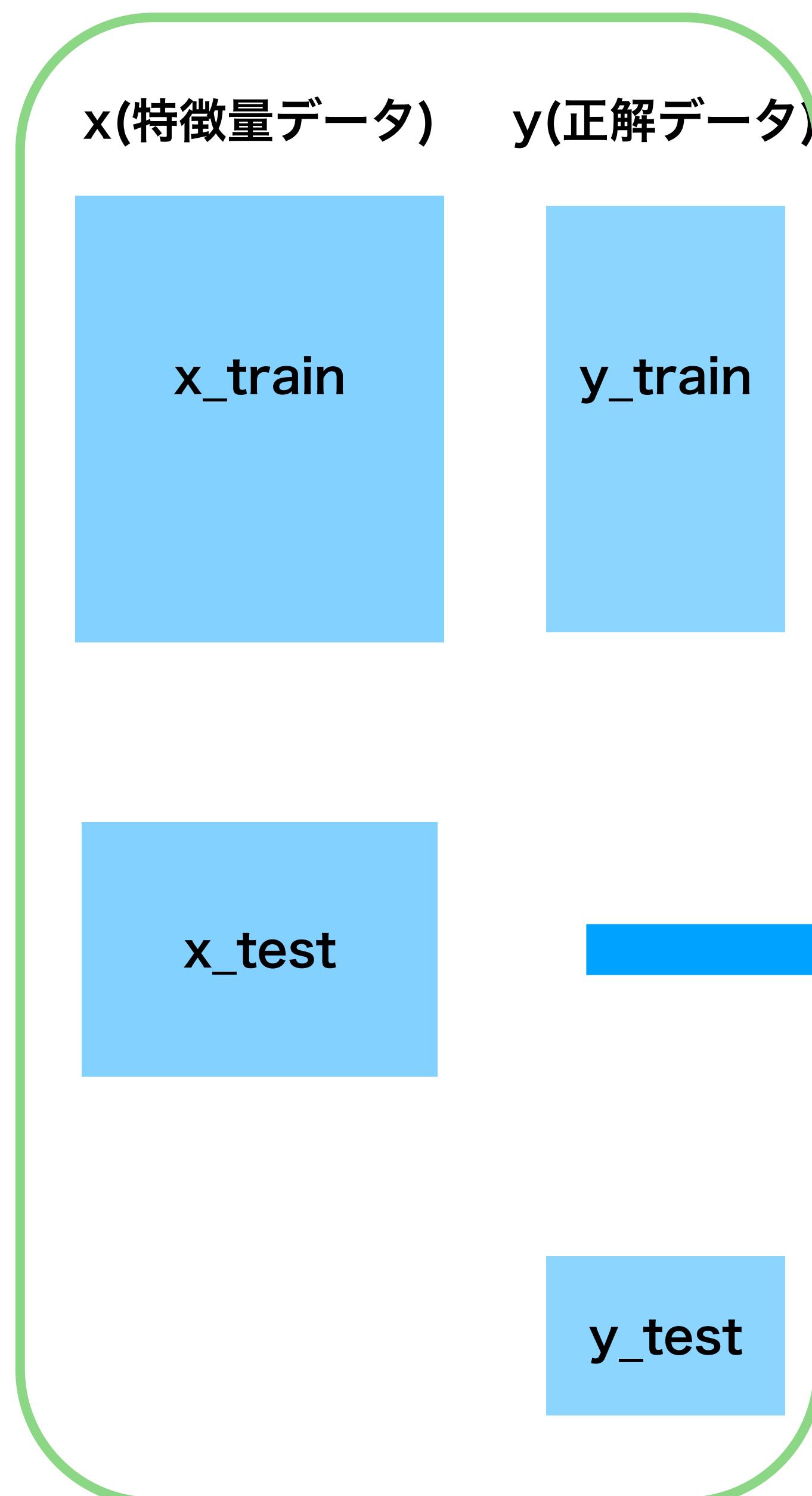
“¥iryoAI¥COVID-NORMAL”になっていることを確認



# 深層学習の流れ

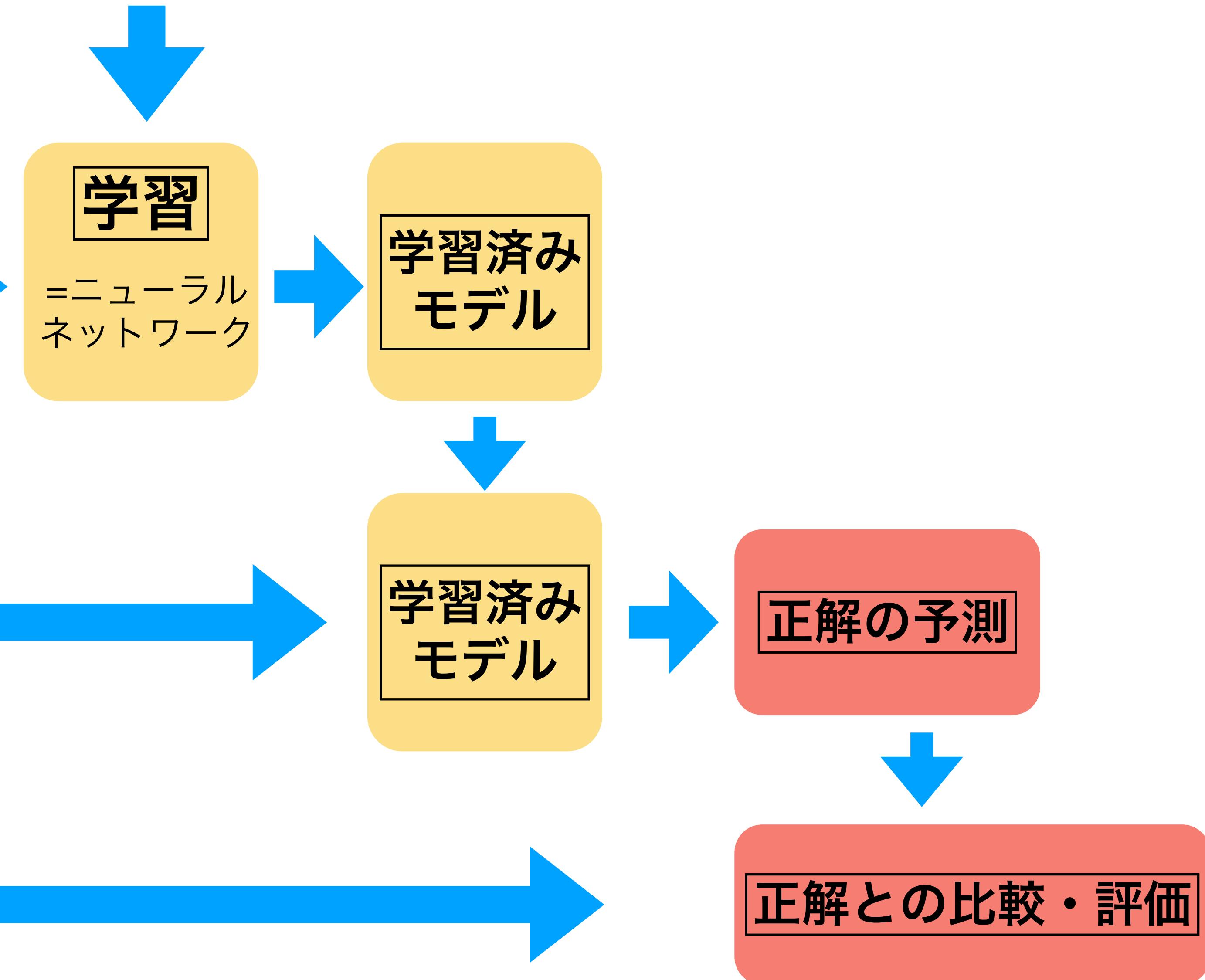


# 画像データ



## 深層学習の流れ

学習モデルの選定



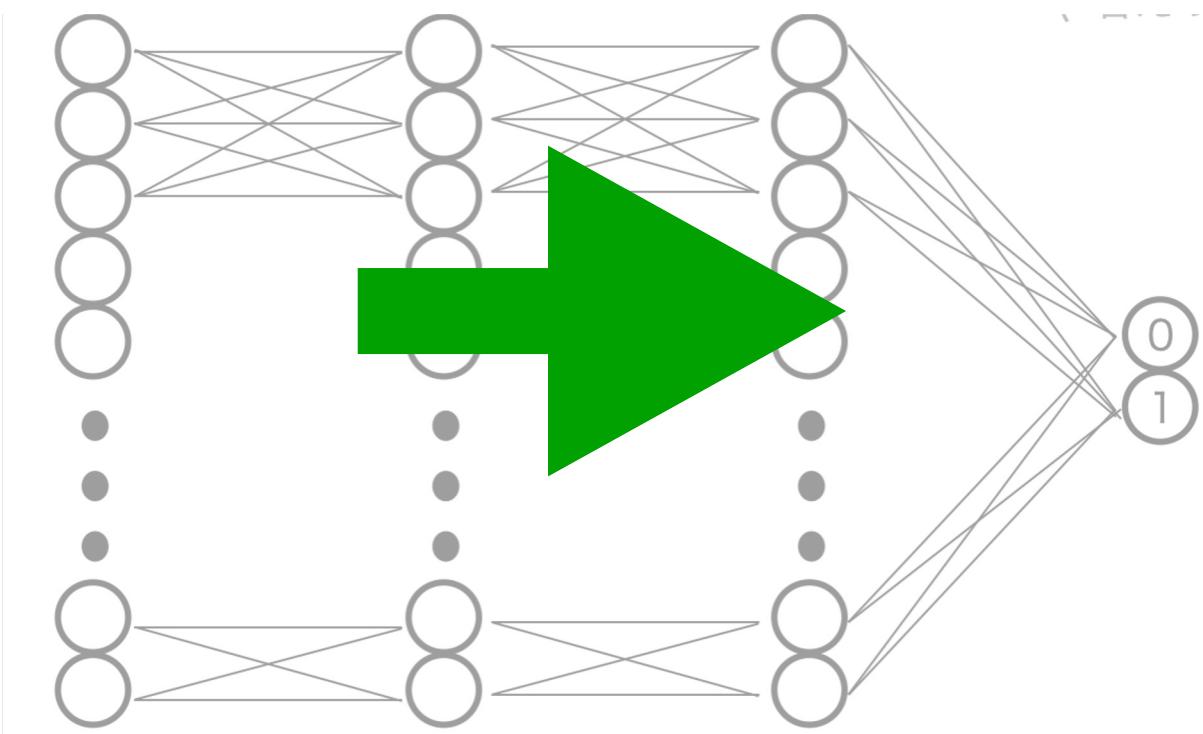
# 肺のレントゲン画像で深層学習

## 学習用データ

A	B	C	D	E	F
	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	教師データー アヤメの種類
1					
2	5.1	3.5	1.4	0.2	0 ヒオウギアヤメ
3	4.9	3	1.4	0.2	0 ヒオウギアヤメ
4	4.7	3.2	1.3	0.2	0 ヒオウギアヤメ
5	4.6	3.1	1.5	0.2	0 ヒオウギアヤメ
6	5	3.6	1.4	0.2	0 ヒオウギアヤメ
7	5.4	3.9	1.7	0.4	0 ヒオウギアヤメ
8	4.6	3.4	1.4	0.3	0 ヒオウギアヤメ
9	5	3.4	1.5	0.2	0 ヒオウギアヤメ
10	4.4	2.9	1.4	0.2	0 ヒオウギアヤメ
11	4.9	3.1	1.5	0.1	0 ヒオウギアヤメ
12	5.4	3.7	1.5	0.2	0 ヒオウギアヤメ
13	4.8	3.4	1.6	0.2	0 ヒオウギアヤメ
14	4.8	3	1.4	0.1	0 ヒオウギアヤメ
15	4.3	3	1.1	0.1	0 ヒオウギアヤメ
16	5.8	4	1.2	0.2	0 ヒオウギアヤメ
17	5.7	4.4	1.5	0.4	0 ヒオウギアヤメ
18	5.4	3.9	1.3	0.4	0 ヒオウギアヤメ
19	5.1	3.5	1.4	0.3	0 ヒオウギアヤメ
20	5.7	3.8	1.7	0.3	0 ヒオウギアヤメ
21	5.1	3.8	1.5	0.3	0 ヒオウギアヤメ
22	5.4	3.4	1.7	0.2	0 ヒオウギアヤメ
23	5.1	3.7	1.5	0.4	0 ヒオウギアヤメ
24	4.6	3.6	1	0.2	0 ヒオウギアヤメ
25	5.1	3.3	1.7	0.5	0 ヒオウギアヤメ
26	4.8	3.4	1.9	0.2	0 ヒオウギアヤメ

がく片の長さと幅  
花びらの長さと幅

## ニューラルネットワーク



## 分類

ヒオウギアヤメorブルーフラッグ

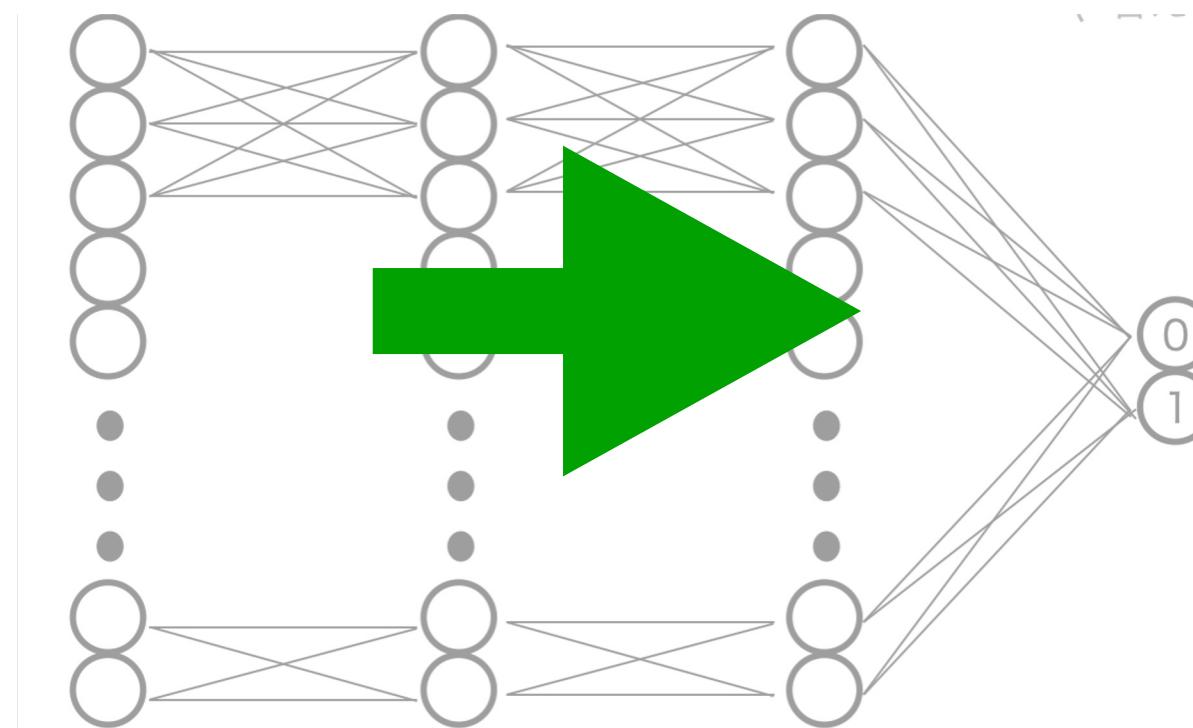
# 肺のレントゲン画像で深層学習

学習用データ

A	B	C	D	E	F
	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	教師データー アヤメの種類
1	5.1	3.5	1.4	0.2	0 ヒオウギアヤメ
2	4.9	3	1.4	0.2	0 ヒオウギアヤメ
3	4.7	3.2	1.3	0.2	0 ヒオウギアヤメ
4	4.6	3.1	1.5	0.2	0 ヒオウギアヤメ
5	5	3.6	1.4	0.2	0 ヒオウギアヤメ
6	5.4	3.9	1.7	0.4	0 ヒオウギアヤメ
7	4.6	3.4	1.4	0.3	0 ヒオウギアヤメ
8	5	3.4	1.5	0.2	0 ヒオウギアヤメ
9	4.4	2.9	1.4	0.2	0 ヒオウギアヤメ
10	4.9	3.1	1.5	0.1	0 ヒオウギアヤメ
11	5.4	3.7	1.5	0.2	0 ヒオウギアヤメ
12	4.8	3.4	1.6	0.2	0 ヒオウギアヤメ
13	4.8	3	1.4	0.1	0 ヒオウギアヤメ
14	4.3	3	1.1	0.1	0 ヒオウギアヤメ
15	5.8	4	1.2	0.2	0 ヒオウギアヤメ
16	5.7	4.4	1.5	0.4	0 ヒオウギアヤメ
17	5.4	3.9	1.3	0.4	0 ヒオウギアヤメ
18	5.1	3.5	1.4	0.3	0 ヒオウギアヤメ
19	5.7	3.8	1.7	0.3	0 ヒオウギアヤメ
20	5.1	3.8	1.5	0.3	0 ヒオウギアヤメ
21	5.4	3.4	1.7	0.2	0 ヒオウギアヤメ
22	5.1	3.7	1.5	0.4	0 ヒオウギアヤメ
23	4.6	3.6	1	0.2	0 ヒオウギアヤメ
24	5.1	3.3	1.7	0.5	0 ヒオウギアヤメ
25	4.8	3.4	1.9	0.2	0 ヒオウギアヤメ
26					

がく片の長さと幅  
花びらの長さと幅

ニューラルネットワーク

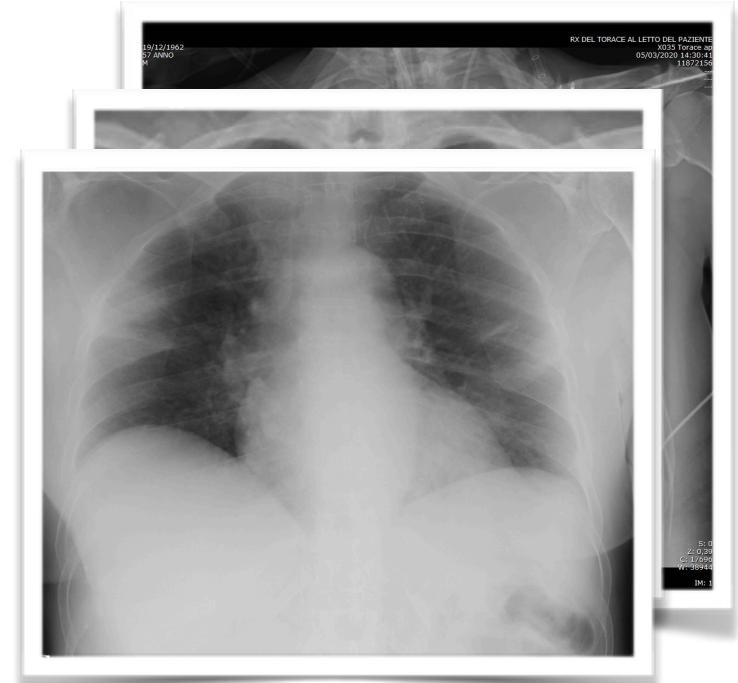


分類

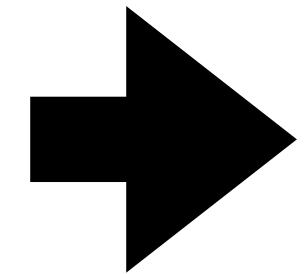
ヒオウギアヤメorブルーフラッグ

デジタル画像

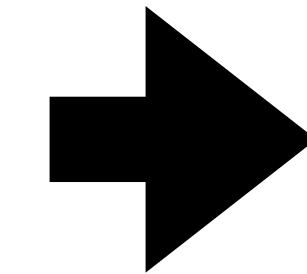
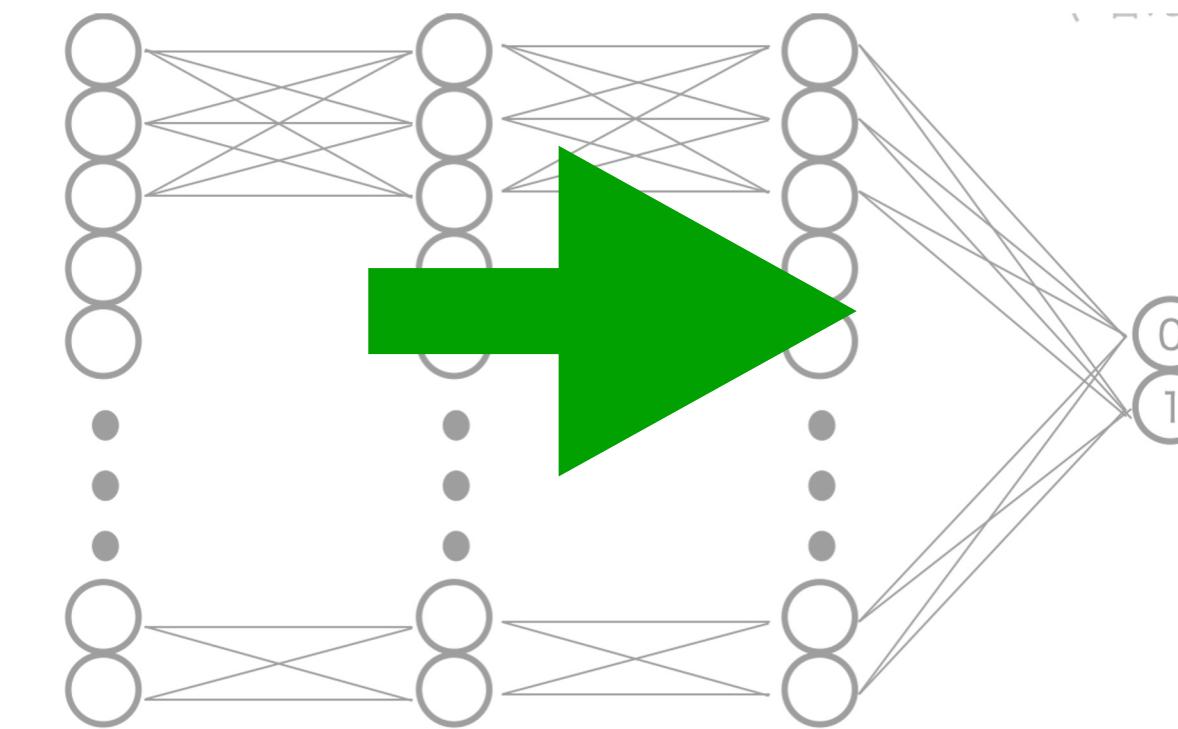
肺炎



健康

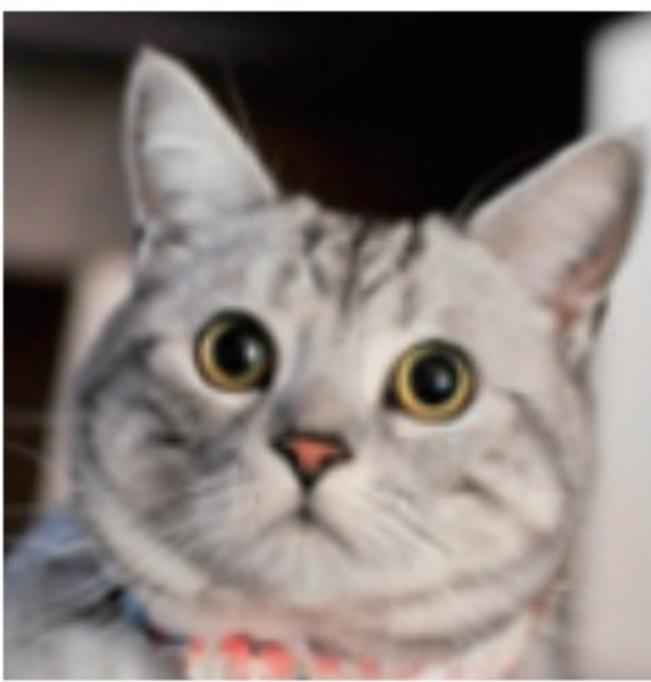


ニューラルネットワーク



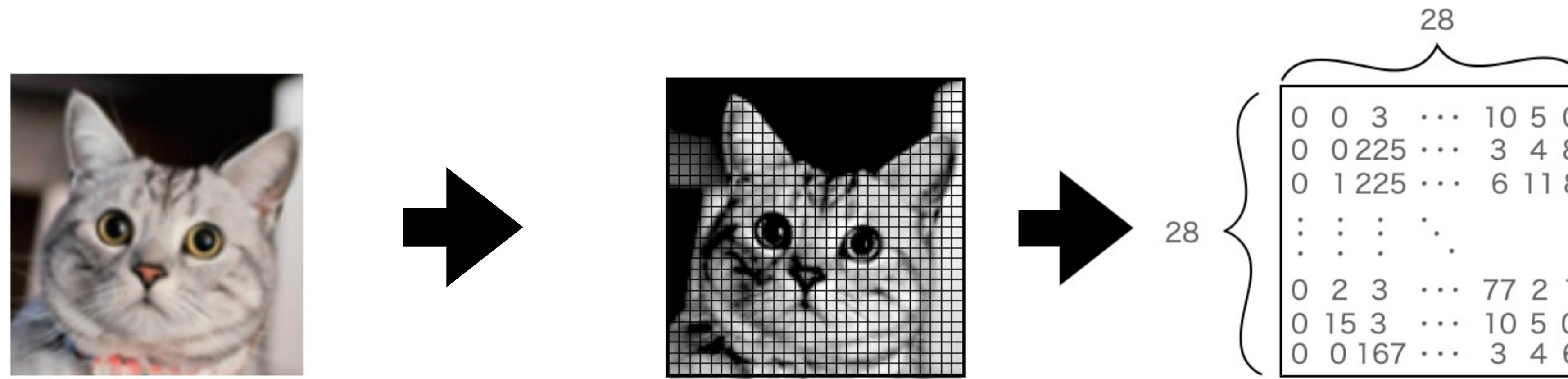
肺炎 or 健康

画像は何が特徴量なのか



# 画像は何が特徴量なのか

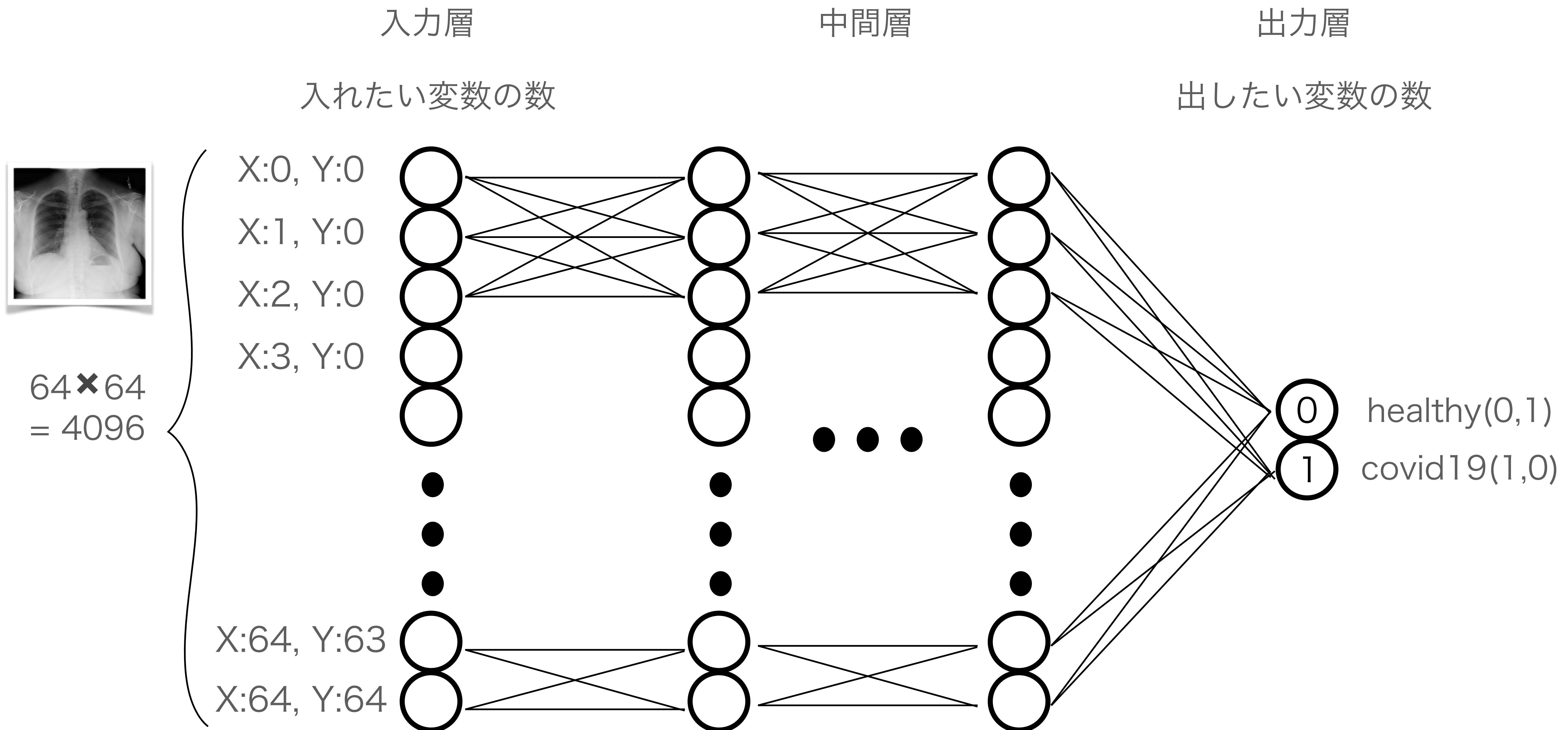
デジタル画像は数字で置き換えることが可能



白黒の濃さを1~255で表示

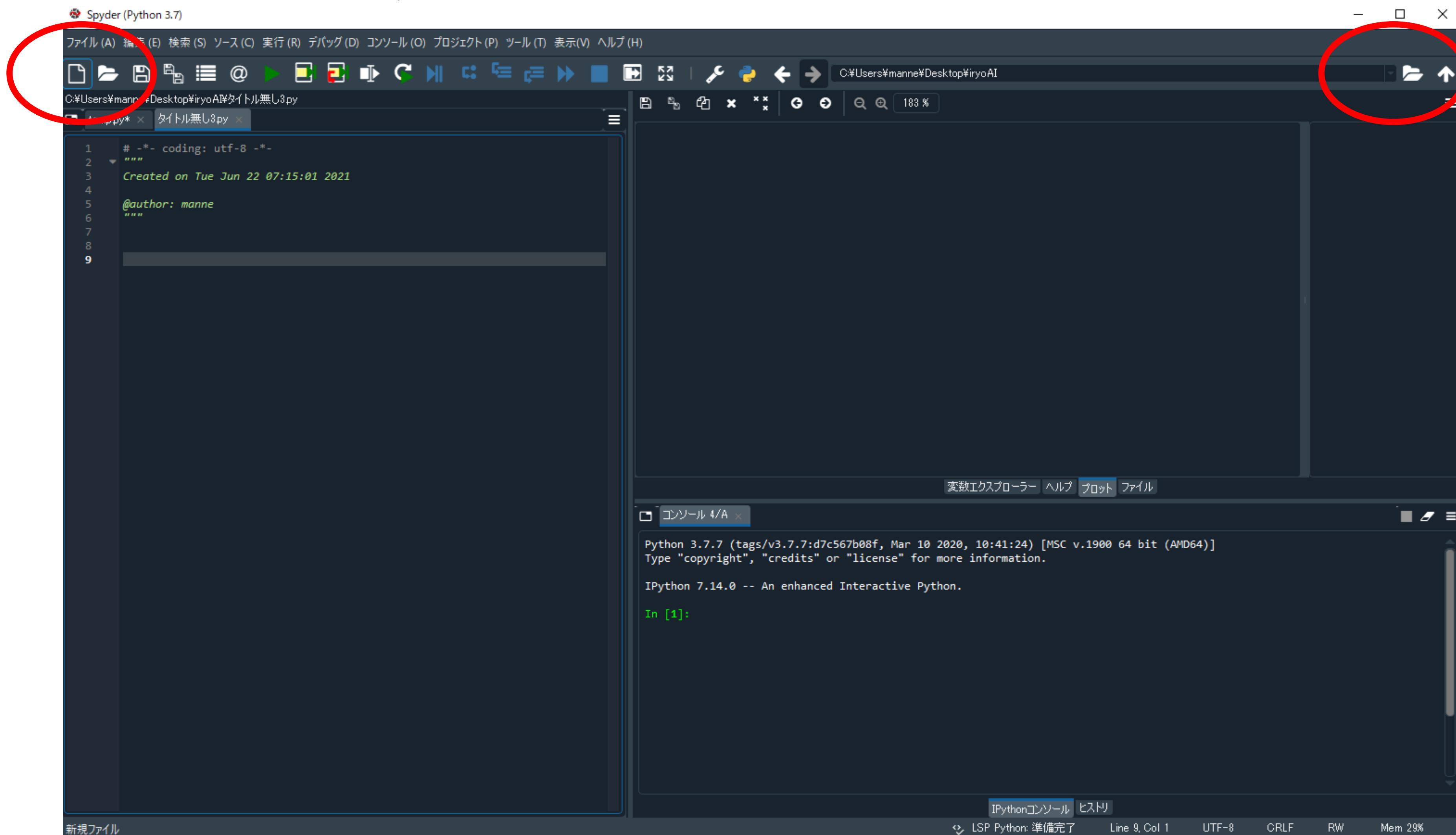
28×28のマス(ピクセルと言います)に色の濃さの数字を当てはめて画像になっている。

# 入力層には各マスの色情報の数値が変数になる



# まず演習の準備をしましょう

- ・新規ファイル作成→iryoAIのディレクトリで”enshu5.py”
- ・作業場所をiryoAIに設定



# ライブラリのインストールとデータの読み込み・加工

## 5.txtの”1)ライブラリのインストール”をコピーする

```
import os
from os import listdir
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['font.family'] ='sans-serif'
rcParams['font.sans-serif'] = ['Hiragino Maru Gothic Pro', 'Yu Gothic', 'Meirio']
# AIのモジュールを取り込む
from keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Flatten,Dense,Dropout

# 以下のコードは本来必要ないが、anacondaでのOMP Abort エラーを防ぐために入れた
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

一度に実行（内容は前回とほぼ一緒です）

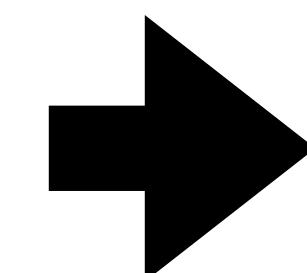
# ライブラリのインストールとデータの読み込み・加工

# やりたいこと①

肺炎



# 健康



# 画像を数値データのNumpy配列に置き換える

```
[[0.47843137] [0]
[0.4745098 ] [0]
[0.49803922] [1]
...
[0.44313725] [1]
[0.45098039] [0]
[0.48235294]] [0]
[0] [0]
...
[[0.02745098] [1]
[0.02745098] [1]
[0.01960784] [1]
...
[0.01176471] [0]
[0.03529412] [0]
[0.04705882]] [1]
[0] [0]
[[0.02745098] [0]
[0.02745098] [0]
[0.02352941] [1]
...
[0.00392157] [1]
[0.03529412] [0]
[0.04705882]] [1]
[1]
[[0.02745098] [0]
[0.02745098] [0]
[0.01960784] [1]
...
[0. ] [1]
[0.03529412] [0]
[0.04313725]] [1]
```

# image\_train (学習用データの説明変数)

labels\_train  
(学習用データの目的変数)

# ライブラリのインストールとデータの読み込み・加工

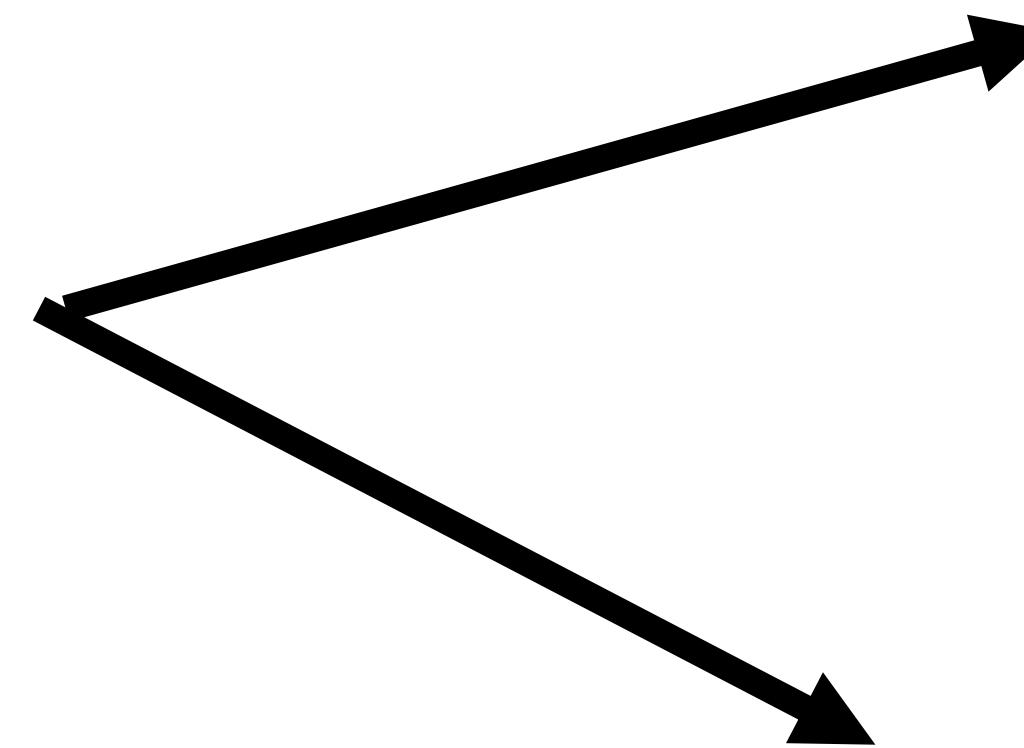
## やりたいこと②

シャッフルして学習用とテスト用に分けたい

肺炎



健康

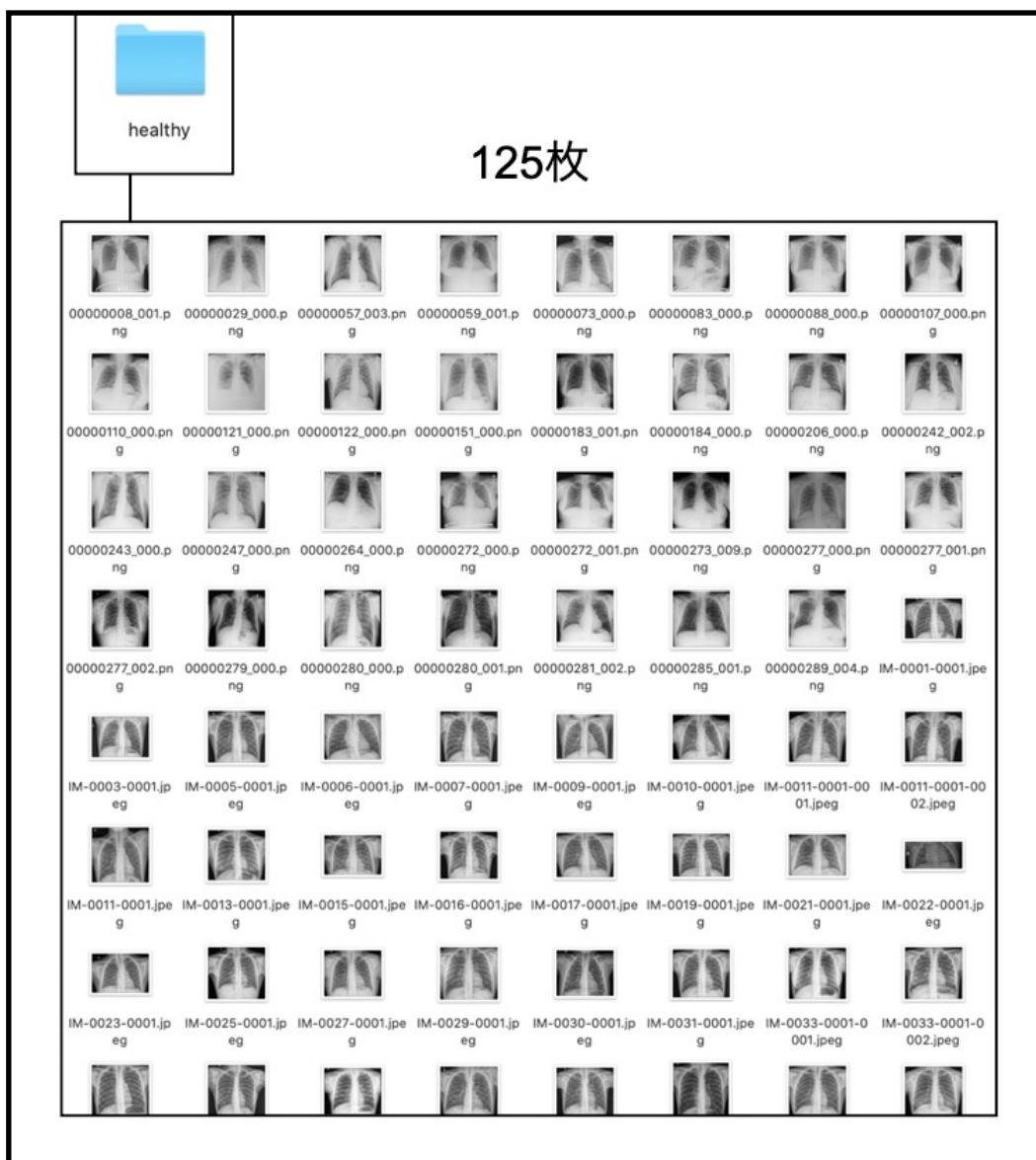


学習用データ  
(肺炎1, 健康2, 肺炎3, ...)

テスト用データ  
(肺炎2, 健康1, 健康3, ...)

# ライブラリのインストールとデータの読み込み・加工

画像を読み込む



numpy配列にする

```
[[0.47843137]
 [0.4745098 ]
 [0.49803922]
 ...
 [0.44313725]
 [0.45098039]
 [0.48235294]]  

...
[[0.02745098]
 [0.02745098]
 [0.01960784]
 ...
 [0.01176471]
 [0.03529412]
 [0.04705882]]  

[[0.02745098]
 [0.02745098]
 [0.02352941]
 ...
 [0.00392157]
 [0.03529412]
 [0.04705882]]  

[[0.02745098]
 [0.02745098]
 [0.01960784]
 ...
 [0. ]
 [0.03529412]
 [0.04313725]]]
```

## # 2)の最初の 4 行をコピーしよう

test.jpgがファイルの保存場所と同じ場所にあることを確認 !!

In

```
test = load_img('./test.jpg', color_mode='rgb')
plt.imshow(test)
plt.show()
print(test)
```

'./test.jp'

“./”は「今いる場所の」という意味

## # 2)の最初の4行をコピーしよう

test.jpgがファイルの保存場所と同じ場所にあることを確認 !!

In

```
test = load_img('./test.jpg', color_mode='rgb')
plt.imshow(test)
plt.show()
print(test)
```

'./test.jp'

“./”は「今いる場所の」という意味

Out



RGBという形式で  
読み込んでいる

サイズは2316×3088

## 次の4行をコピーしよう

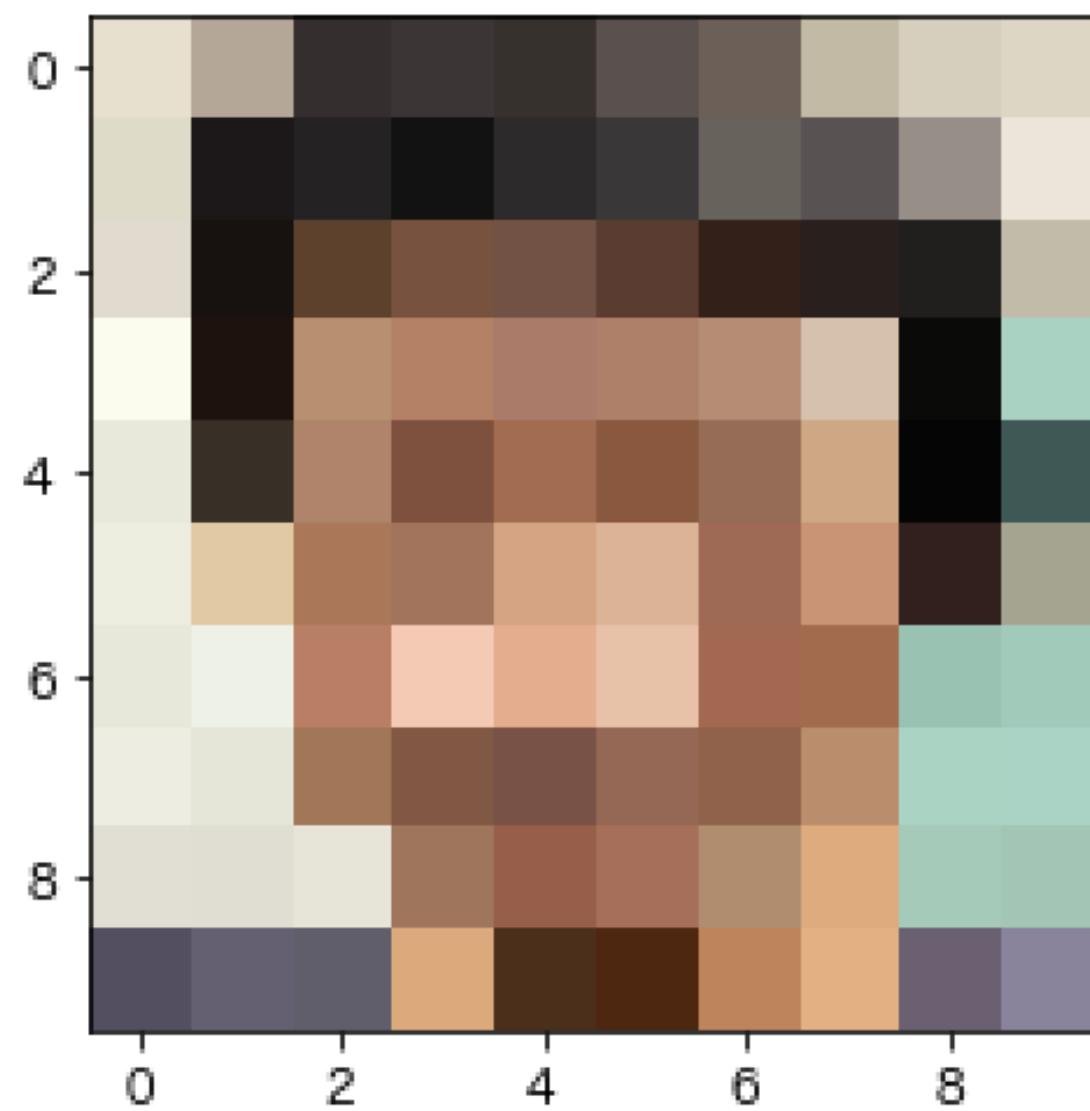
```
test2 = load_img('./test.jpg', color_mode='rgb',target_size=(10,10))
plt.imshow(test2)
plt.show()
print(test2)
```

10×10でRGBで読み込む  
画像の表示  
配列へ変換  
サイズを表示

## 画像を荒くする(10×10)

```
test2 = load_img('./test.jpg', color_mode='rgb',target_size=(10,10))
plt.imshow(test2)
plt.show()
print(test2)
```

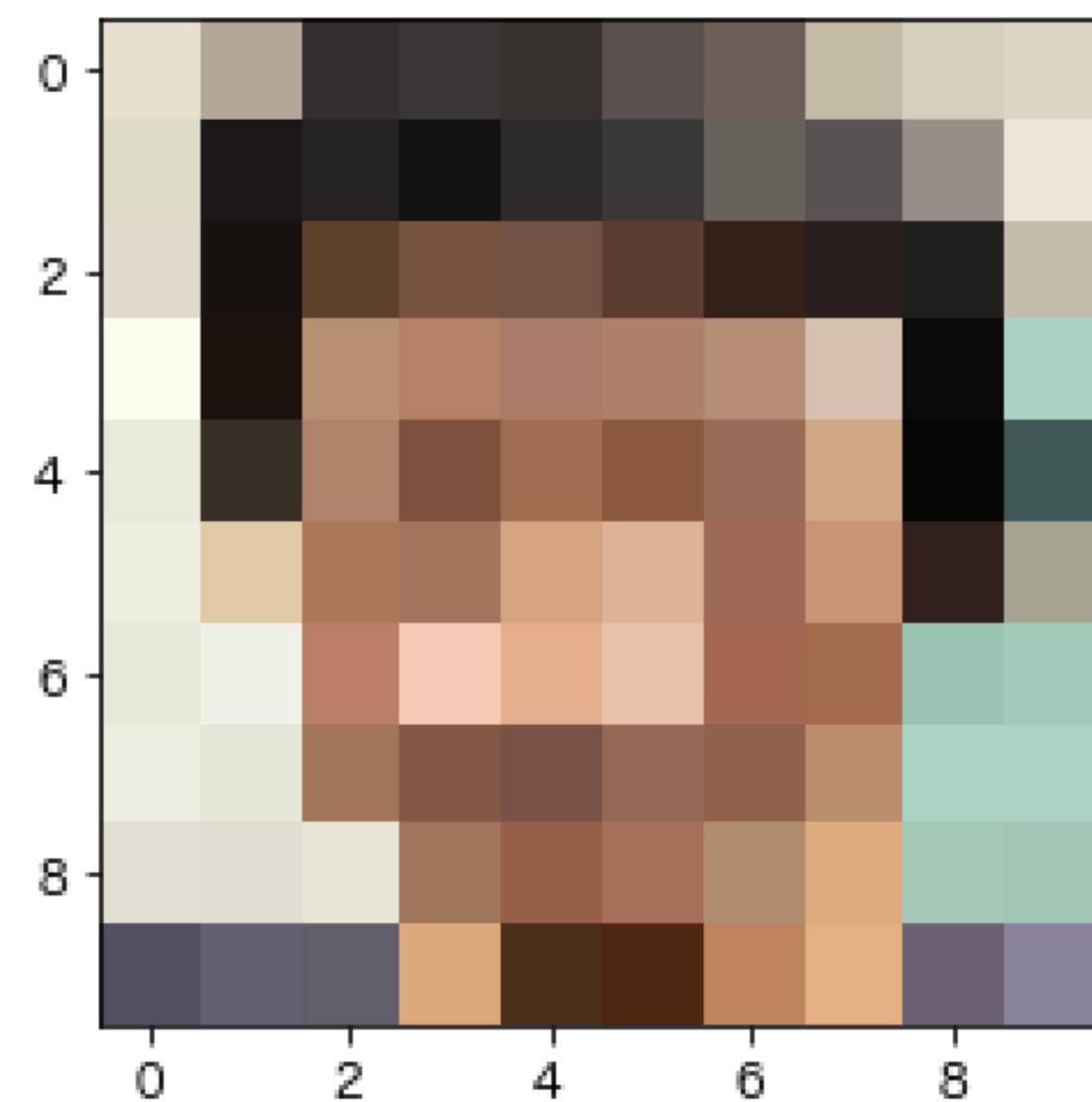
10×10でRGBで読み込む  
画像の表示  
配列へ変換  
サイズを表示



## 次の3行をコピーしよう

```
test2_img = img_to_array(test2)
print(test2_img)
print(test2_img.shape)
```

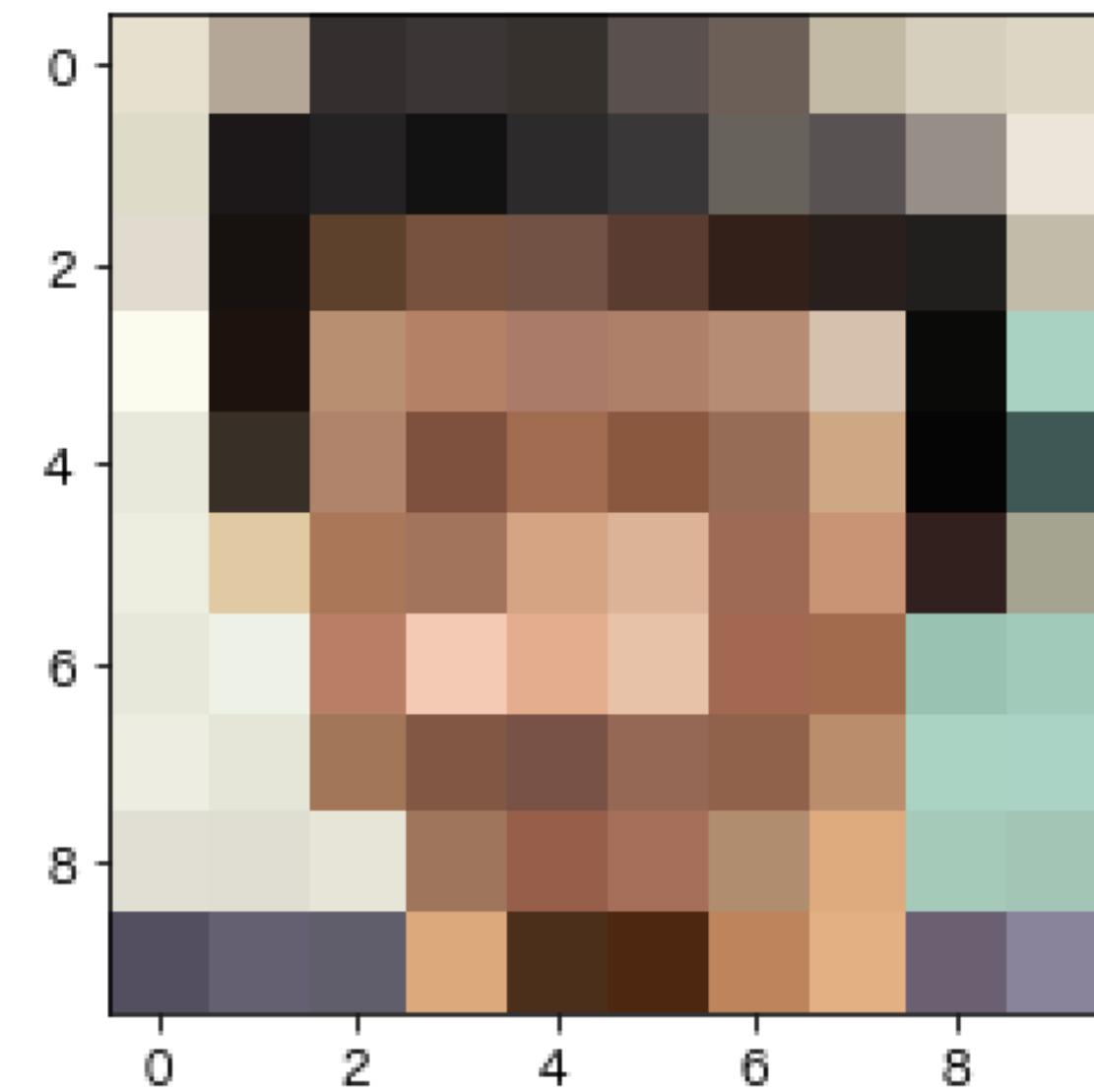
画像データを配列に変換  
配列の表示  
形状を表示



## 配列に変換

```
test2_img = img_to_array(test2)
print(test2_img)
print(test2_img.shape)
```

画像データを配列に変換  
配列の表示  
形状を表示



```
[[[230. 224. 208.]
 [179. 167. 153.]
 [ 52. 46. 46.]
 [ 58. 54. 53.]
 [ 54. 49. 46.]
 [ 88. 80. 77.]
 [106. 95. 89.]
 [194. 186. 167.]
 [213. 207. 191.]
 [220. 214. 198.]]
```

(10, 10, 3)

## 次の7行をコピーしよう

In

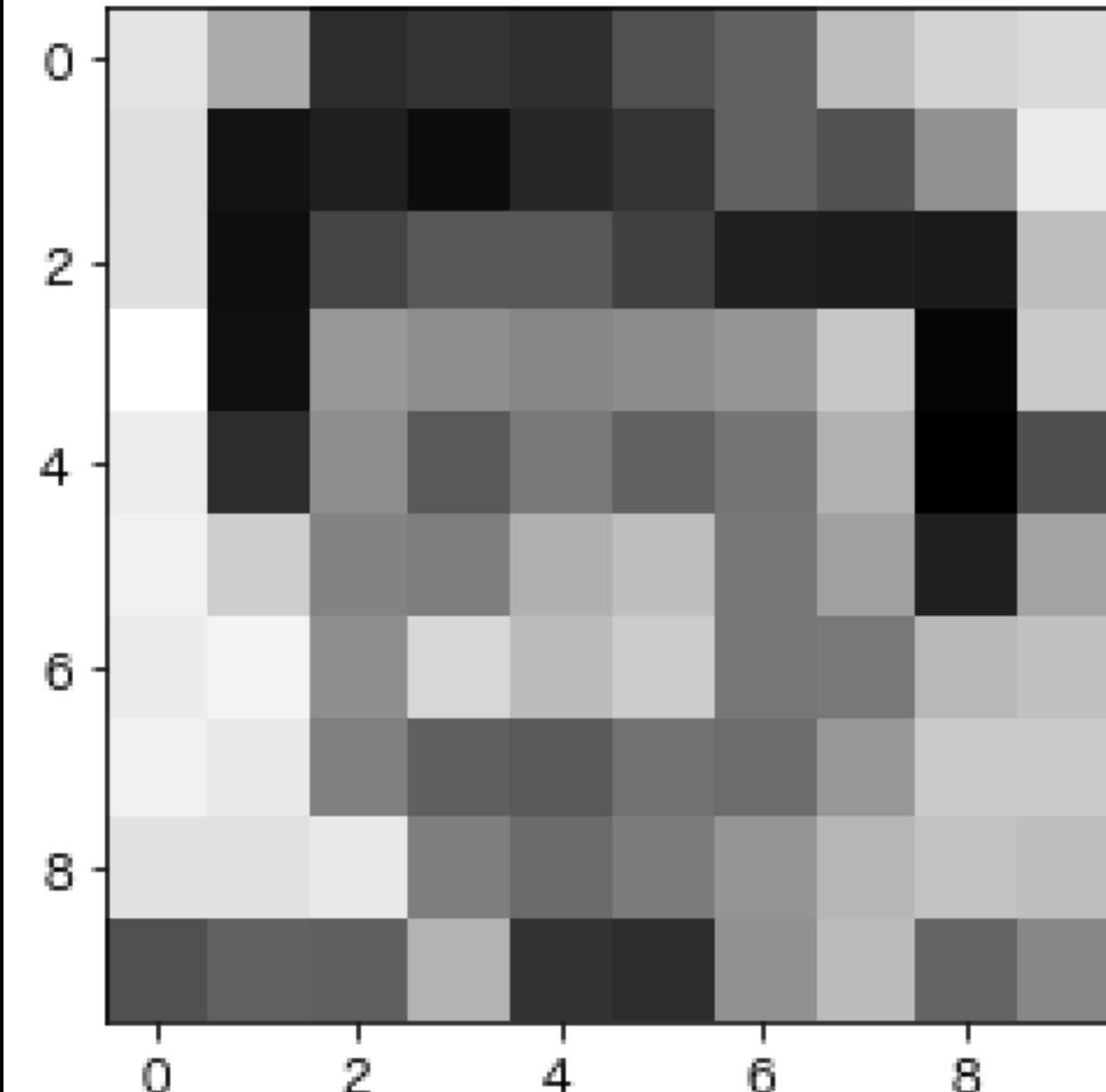
```
test3 = load_img('./test.jpg',color_mode='grayscale',target_size=(10,10))
plt.imshow(test3)
plt.gray()
plt.show()
print(test3)
test3_img = img_to_array(test3)
print(test3_img)
print(test3_img.shape)
```

## 次の7行をコピーしよう

In

```
test3 = load_img('./test.jpg',color_mode='grayscale',target_size=(10,10))
plt.imshow(test3)
plt.gray()
plt.show()
print(test3)
test3_img = img_to_array(test3)
print(test3_img)
print(test3_img.shape)
```

Out

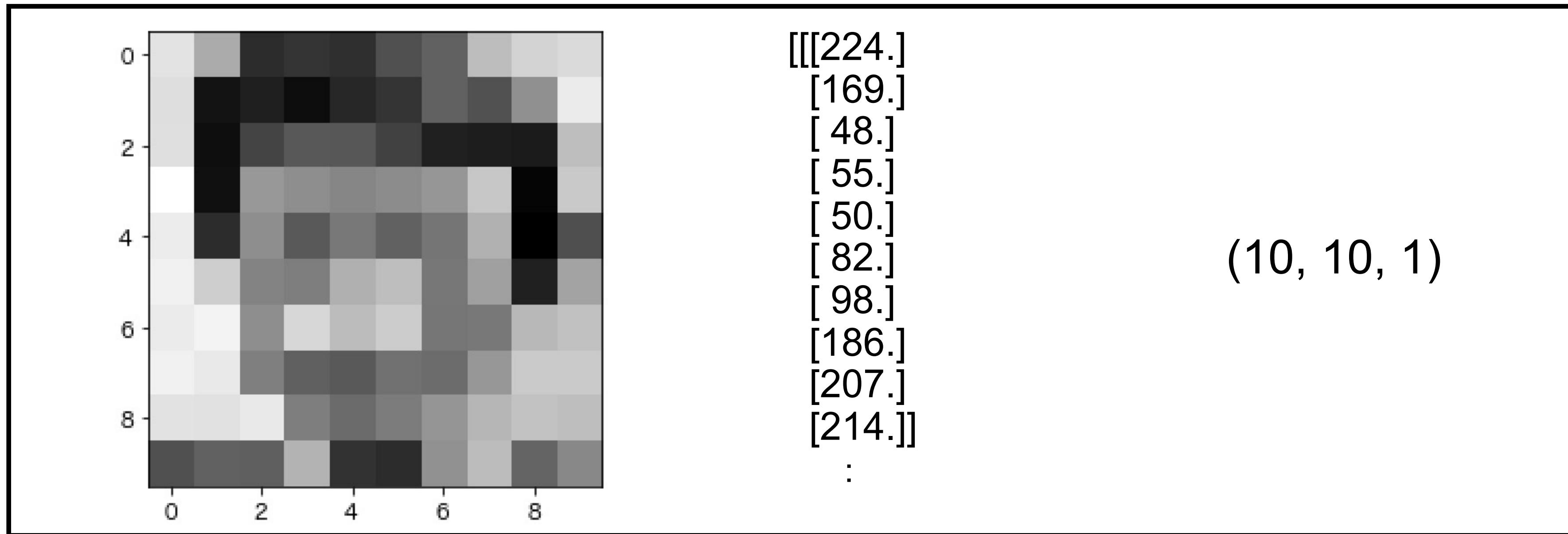


<PIL.Image.Image image mode=L size=10x10 at 0x7FEF20BF9C10>

10×10の解像度に変更された  
(さらに荒くなっている)

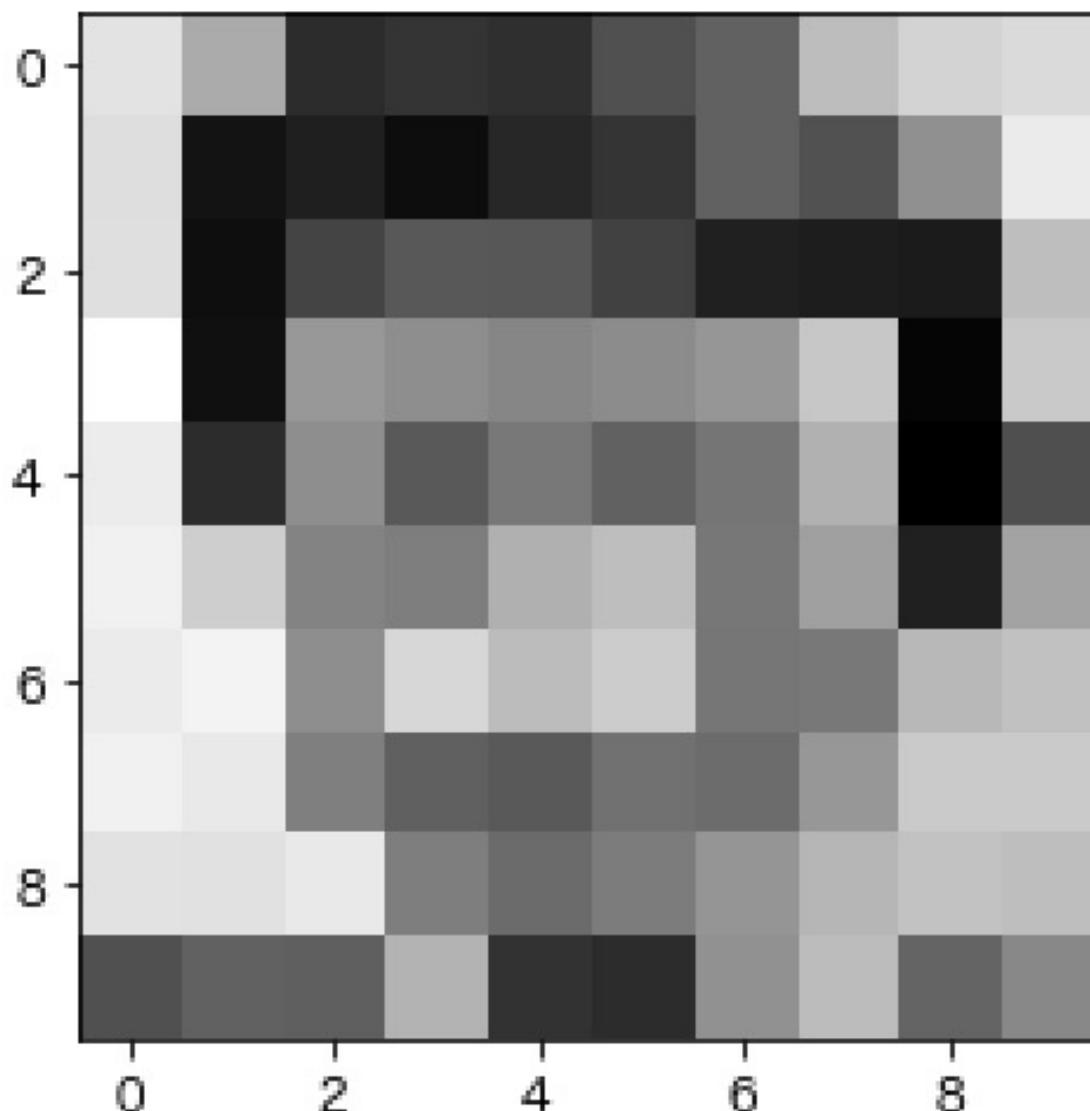
グレースケール(grayscale)は白黒

# (10, 10, 1)の1は何を表しているか



一番内側の[ ]が1つのピクセル  
2つ目の[ ]で1行(10行)  
3つめが全体

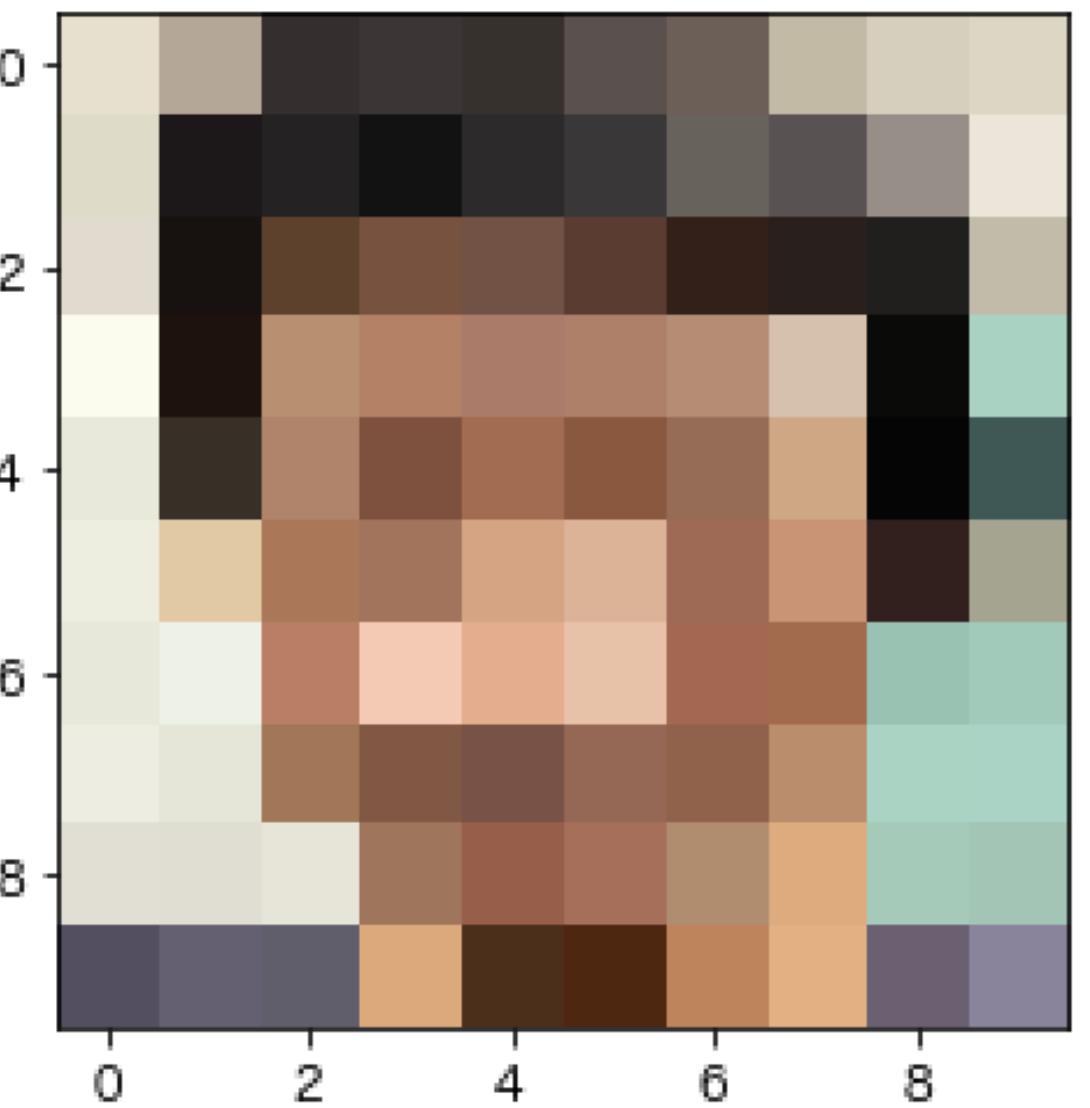
# (10, 10, 1)の1は何を表しているか



```
[[[224.  
169.  
48.  
55.  
50.  
82.  
98.  
186.  
207.  
214.]]  
:  
:
```

(10, 10, 1)

一番内側の[224.]  
で白黒を表現



```
[[[230. 224. 208.  
179. 167. 153.  
52. 46. 46.  
58. 54. 53.  
54. 49. 46.  
88. 80. 77.  
106. 95. 89.  
194. 186. 167.  
213. 207. 191.  
220. 214. 198.]]  
:  
:
```

(10, 10, 3)

赤 緑 青

一番内側の[230. 224. 208.]  
でRGBを表現

# 画像を複数読み込んだ場合は？



# 画像を複数読み込んだ場合は？



```
[ [ [ [224.] [169.] [169.] [169.]  
[169.] [ 48.] [ 48.] [ 48.]  
[ 48.] [ 55.] [ 55.] [ 55.]  
[ 55.] [ 50.] [ 50.] [ 50.]  
[ 50.] [ 82.] [ 82.] [ 82.]  
[ 82.] [ 98.] [ 98.] [ 98.]  
[ 98.] [186.] [186.] [186.]  
[186.] [207.] [207.] [207.]  
[207.] [214.]] [214.]] [214.]]  
[214.]] : : :  
[222. ]]] [222. ]]] [222. ]]] [222. ]]] ]
```

画像を複数読み込んだ配列は(枚数、横のセル、縦のセル、色の数)になる

# 画像の読み込みと加工

#3) 実際の画像を読み込み加工する

#画像ファイル名取得

```
list_healthy = [i for i in listdir('./COVID-NORMAL/healthy') if not i.startswith('.')]  
list_covid19 = [i for i in listdir('./COVID-NORMAL/covid19') if not i.startswith('.')]
```

#画像ファイル数

```
number_of_healty = len(list_healthy)  
number_of_covid19 = len(list_covid19)  
number_of_total = number_of_healty + number_of_covid19
```

#データ保管用numpy行列を定義

```
images_input = np.zeros((number_of_total, 64, 64, 1), dtype=int) #画像 64×64 白黒 0-255整数  
images_train = np.zeros((number_of_total, 64, 64, 1), dtype=float) #画像 64×64 白黒 0-1浮動小数点  
labels_train = np.zeros((number_of_total, 1), dtype=int) #ラベル 0 : 正常 1:COVID19
```

#画像ファイル取り込み

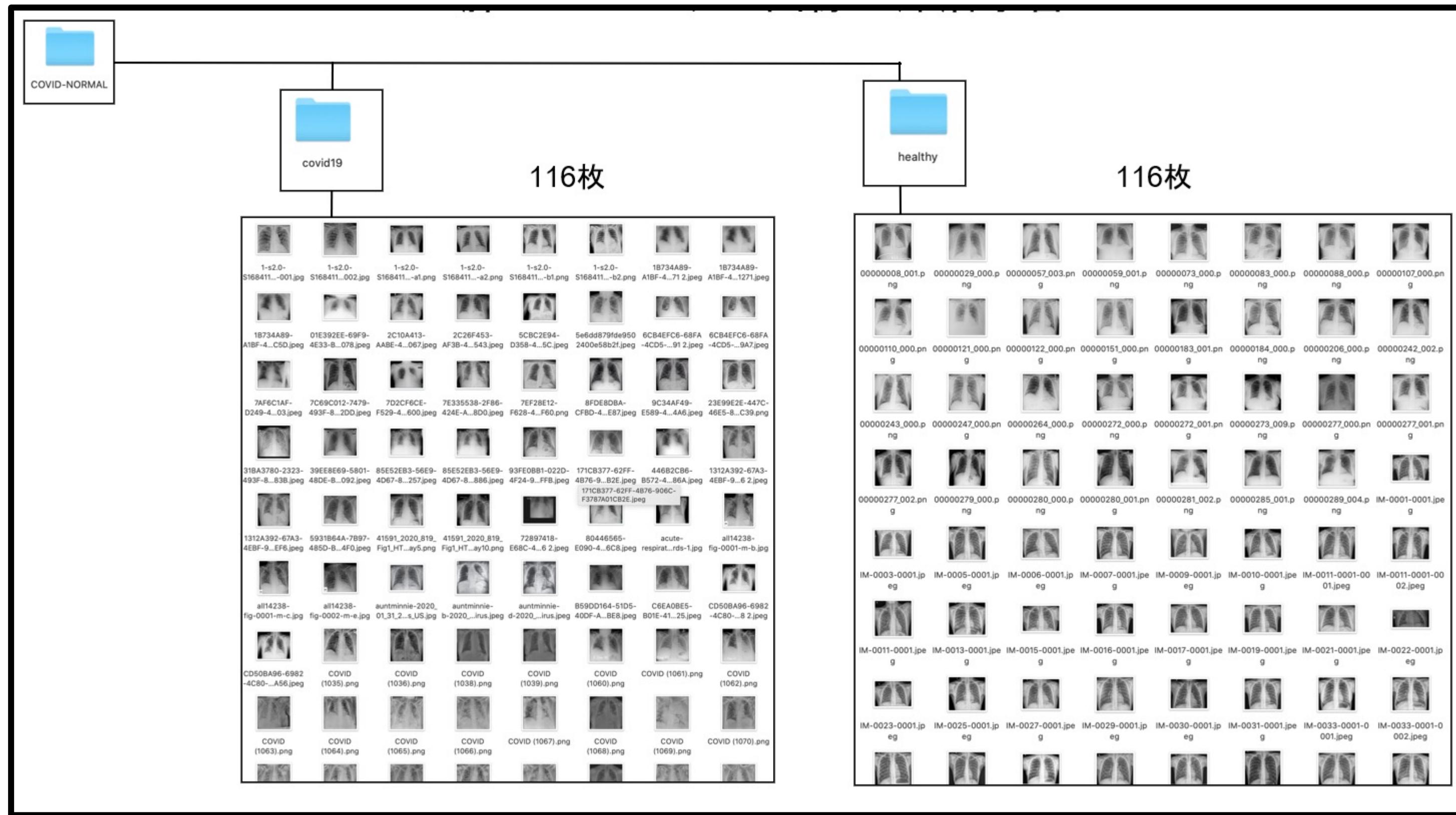
```
for i in range(0, number_of_healty):  
    filepath = './COVID-NORMAL/healthy/%s' %list_healthy[i]  
    temp = load_img(filepath, color_mode='grayscale', target_size=(64,64), interpolation='lanczos' )  
    images_input[i] = img_to_array(temp)  
for i in range(0, number_of_covid19):  
    filepath = './COVID-NORMAL/covid19/%s' %list_covid19[i]  
    temp = load_img(filepath, color_mode='grayscale', target_size=(64,64), interpolation='lanczos' )  
    images_input[i + number_of_healty] = img_to_array(temp)
```

#画像ファイルの並び順をランダムにシャッフル

```
num_list = random.sample(range(250), k=250) # 0 から 249 までの整数乱数のリストを作成  
print(num_list)  
j = 0  
for i in num_list:  
    images_train[j] = images_input[i] /255.0  
    labels_train[j] = 0 if i < number_of_healty else 1  
    j = j + 1
```

今回は詳細を割愛しますが、  
webclassにここの詳しい説明を動画で  
アップおります。

## 前処理のゴール(#3)でやっていること)



232枚の全て読み込んで、(学習+検証用データ) = (232, 64, 64, 1)を作る

健常と肺炎の画像をランダムにシャッフルして、  
image\_train(特徴量データ)とlabels\_train(正解データ)を作成

## #4) データの確認

```
print(images_input)
print(images_train)
print(images_train.shape)
print(labels_train.shape)
```

## #4) データの確認

```
print(images_input)
print(images_train)
print(images_train.shape)
print(labels_train.shape)
```

images\_input  
232枚を  
順に読み込んで  
配列に変換

```
[[[ [ 20]
    [ 10]
    [ 6]
    ...
    [ 9]
    [ 11]
    [ 13]]]
```

```
[[ [ 17]
    [ 33]
    [ 61]
    ...
    [ 2]
    [ 5]
    [ 9]]]
```

```
[[ [ 78]
    [ 109]
    [ 119]
    ...
    [ 35]
    [ 36]
    [ 38]]]]
```

images\_train  
順番をシャッフル  
して255で割って  
0～1に変換  
(色の濃さが0～  
255)

```
[[[[ 0.14901961]
    [ 0.1372549 ]
    [ 0.10980392]
    ...
    [ 0.02745098]
    [ 0.04705882]
    [ 0.06666667]]]
```

```
[[ [ 0.12941176]
    [ 0.10196078]
    [ 0.09803922]
    ...
    [ 0.11372549]
    [ 0.04313725]
    [ 0.04313725]]]
```

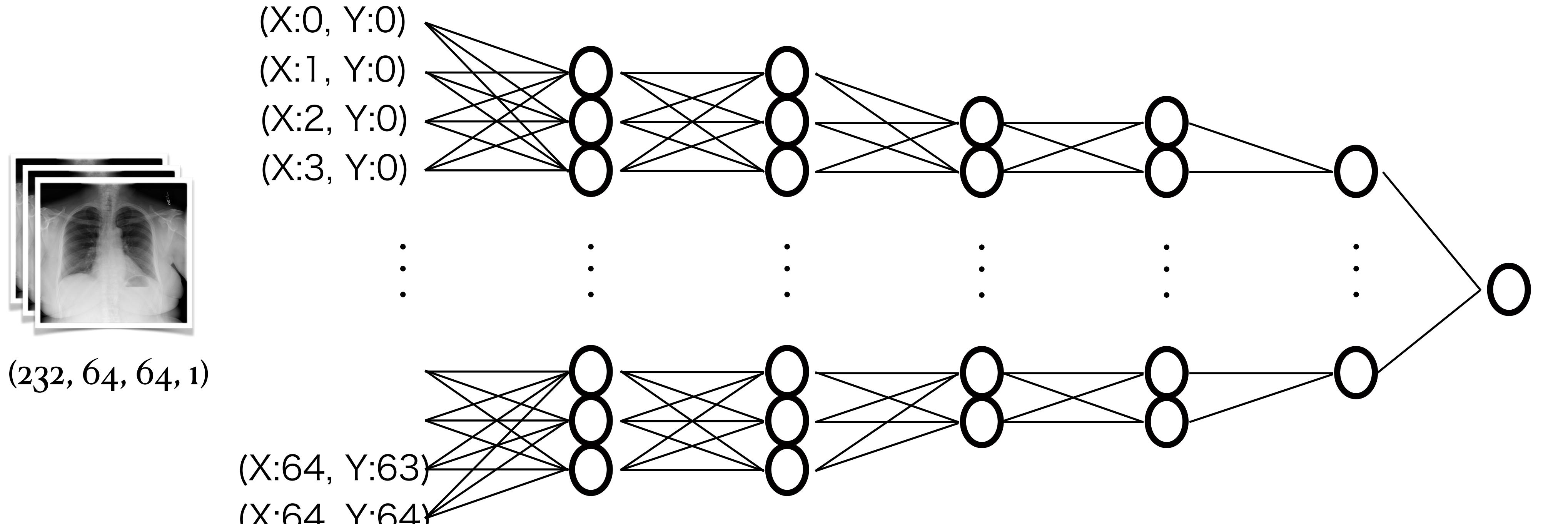
```
[[ [ 0.09411765]
    [ 0.17254902]
    [ 0.28627451]
    ...
    [ 0.10588235]
    [ 0.01176471]
    [ 0.02745098]]]]
```

print(images\_train.shape)  
(232, 64, 64, 1)

print(labels\_train.shape)  
(232, 1)

健常が0、肺炎が1

# 学習モデルの作成



入力データ

4096個 (ニューロンの数)	1層目	2層目	3層目	4層目	5層目	出力層
	512個	256個	128個	64個	32個	1個
Flatten	Dense	Dense	Dense	Dense	Dense	Dense
	(Dropout)	(Dropout)	(Dropout)			

# 学習モデルの作成

ニューラルネットワークを作成していく

(モデル名) = Sequential()

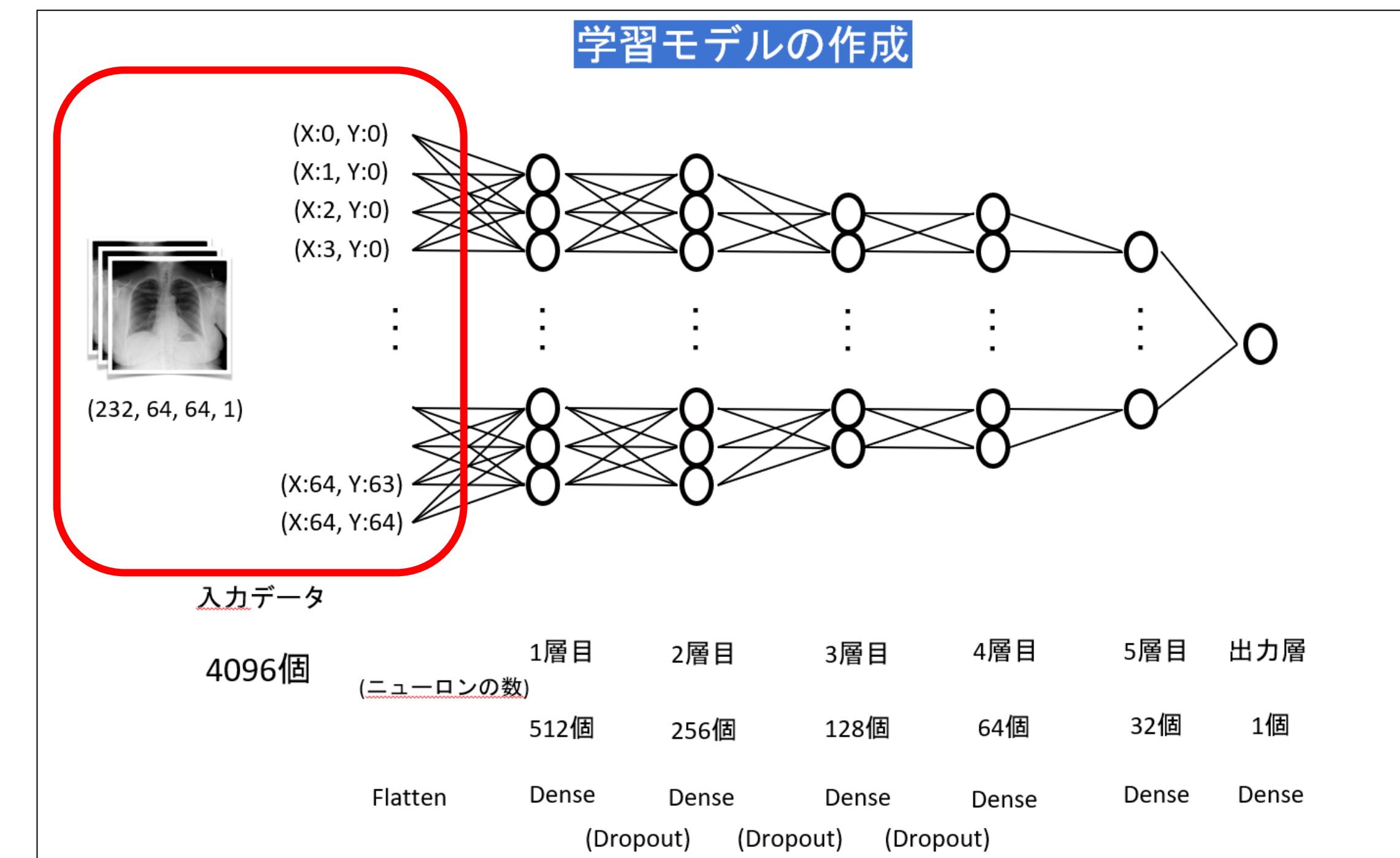
```
model = Sequential()
model.add(Flatten(input_shape=(64, 64, 1)))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
model.summary()
```

# 学習モデルの作成

```
model.add(Flatten(input_shape=(64, 64, 1)))
```

Flatten層は、入力データであるinput\_shapeを1次元ベクトルに変換します  
今回は(64, 64, 1)の3次元ベクトルなので、 $64 \times 64 \times 1 = 4096$ の1次元ベクトル  
に変換します。

(64, 64, 1 ) → (4096, )

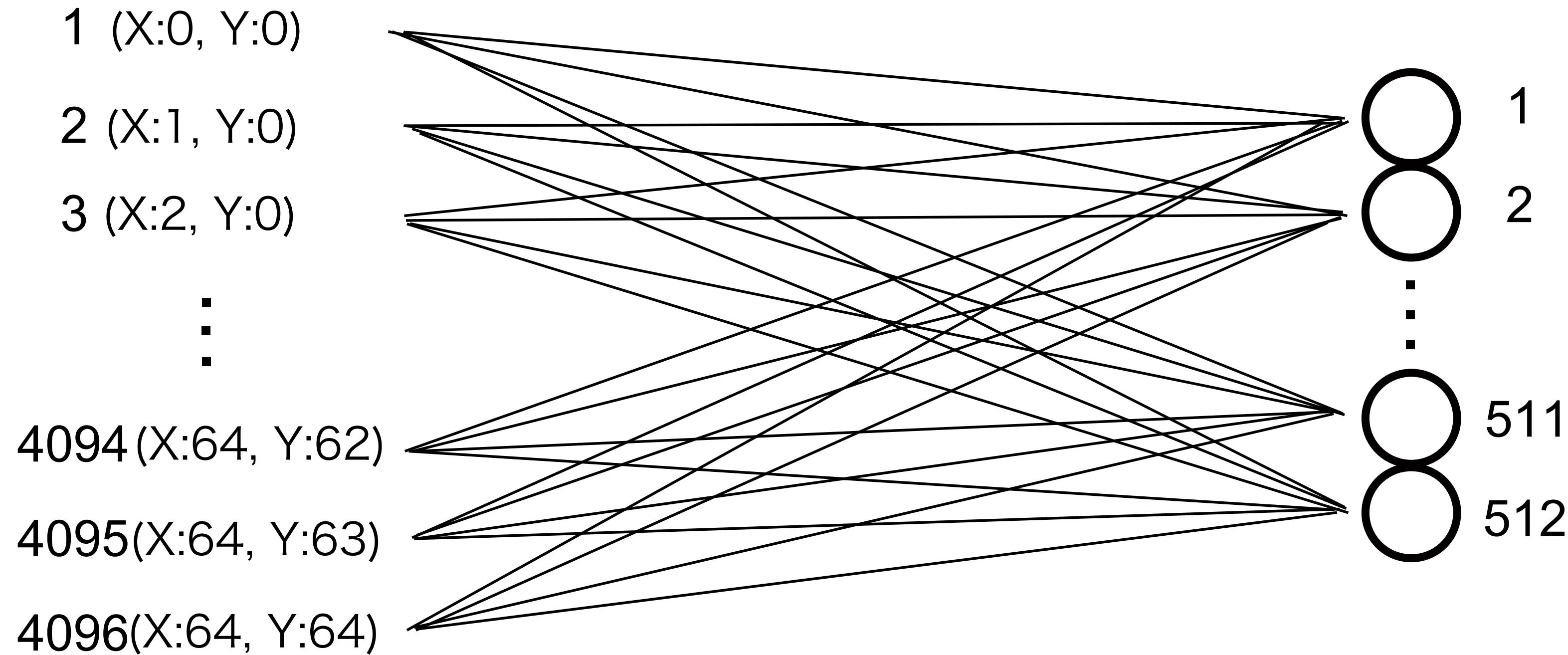


# 学習モデルの作成

ニューラルネットワークを作成していく

(モデル名) = Sequential()

```
model = Sequential()  
model.add(Flatten(input_shape=(64, 64, 1)))  
model.add(Dense(512, activation='relu'))
```

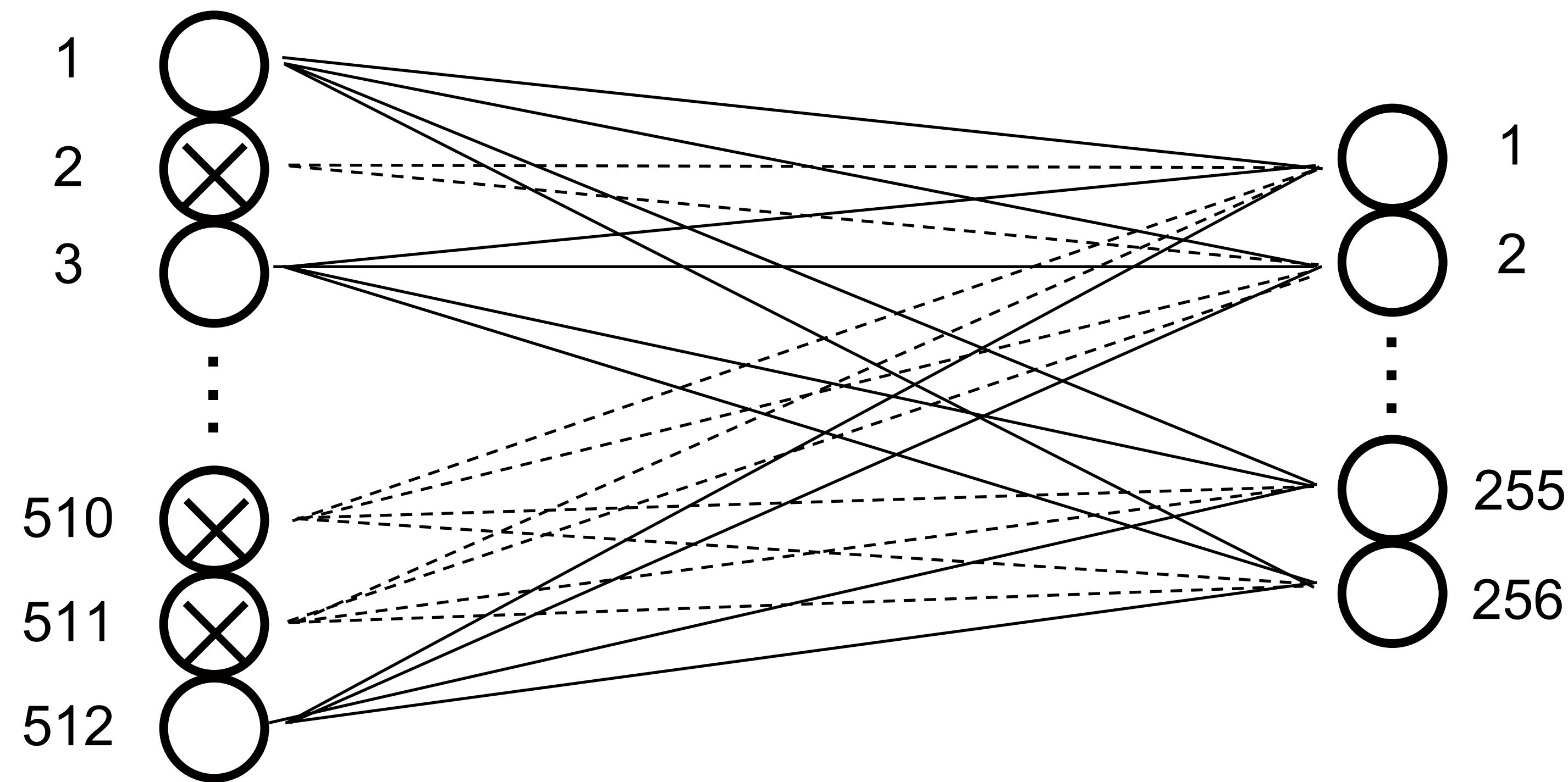


# 学習モデルの作成

```
model = Sequential()  
model.add(Flatten(input_shape=(64, 64, 1)))  
model.add(Dense(512, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(256, activation='relu'))
```

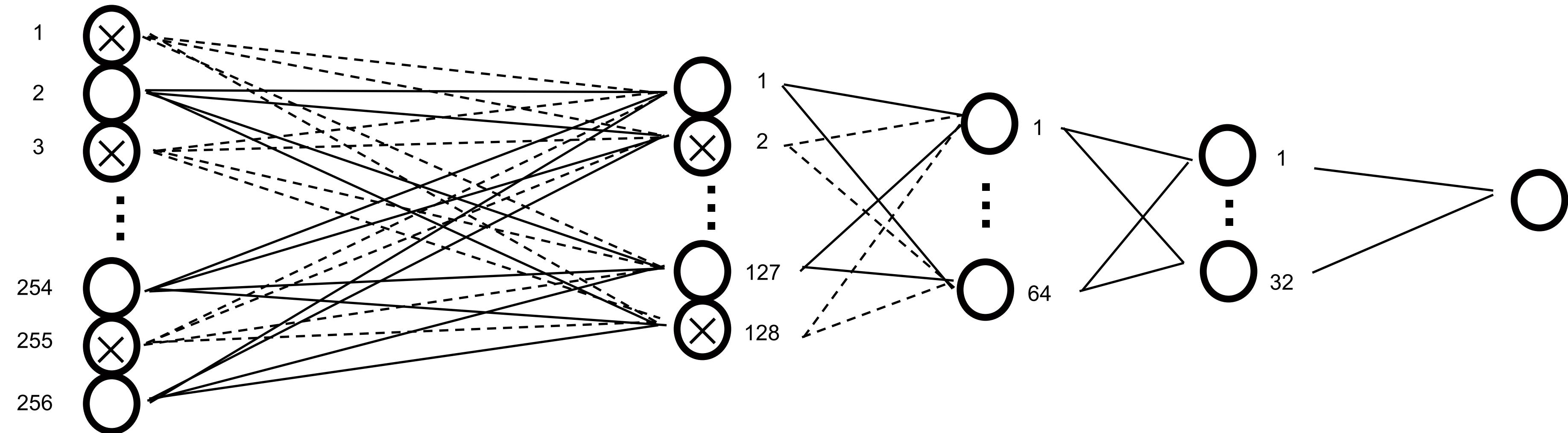
## Dropout :

学習データの特徴を過剰に評価すること(過学習)を防ぐため、学習時に設定した確率で出力を 0 にする手法。推論時(順方向伝播時)には何も行わず、学習時(逆方向伝播時)にのみ行われる。



# 学習モデルの作成

```
model = Sequential()  
model.add(Flatten(input_shape=(64, 64, 1)))  
model.add(Dense(512, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(256, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(128, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(64, activation='relu'))  
model.add(Dense(32, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))
```



# 学習モデルの作成

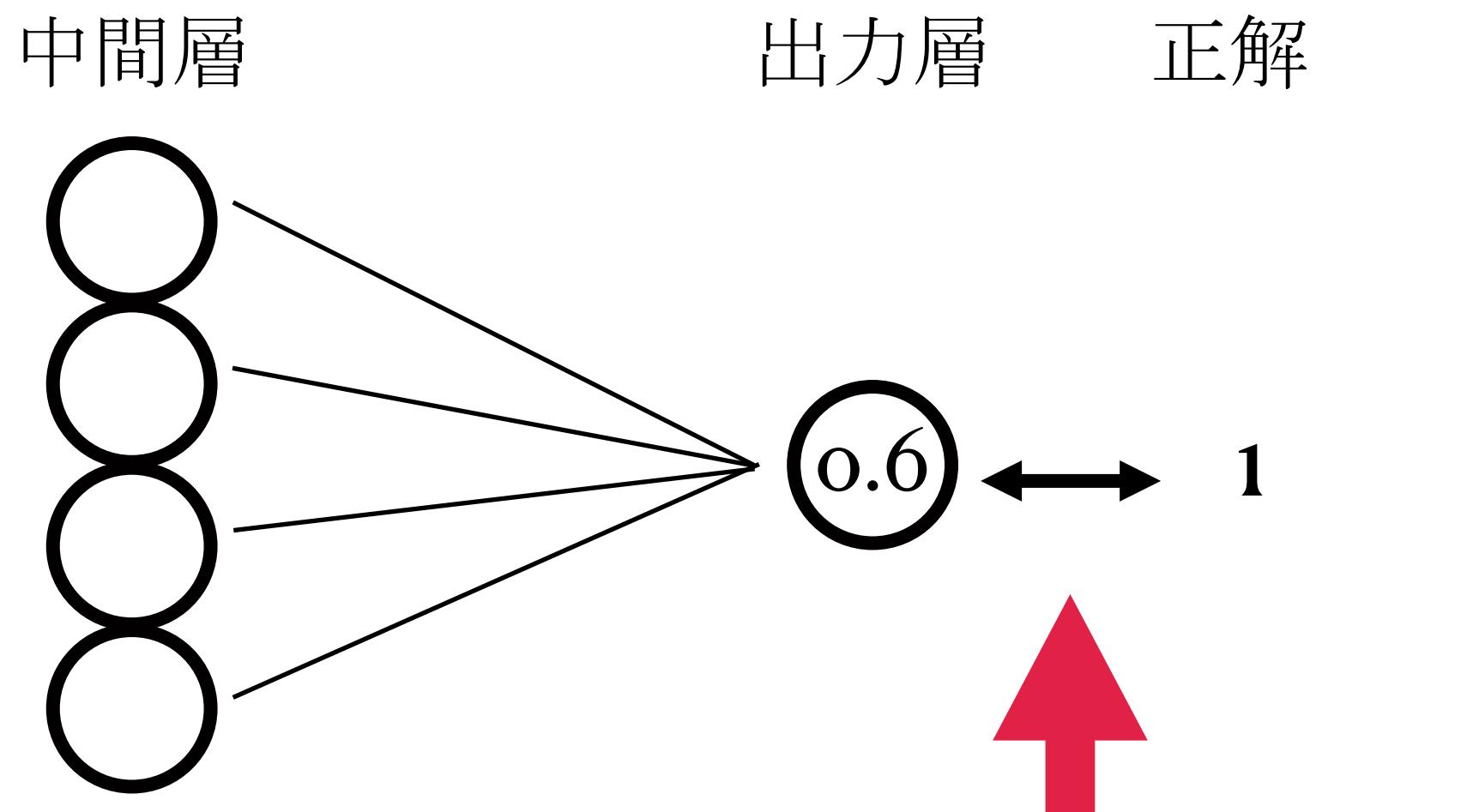
model.compile()で、学習時の評価方法の選択をする

loss = “損失関数”, optimizer = ‘最適化関数’, metrics=[“評価指標”]

今回は2クラス分類なのでbinary\_crossentropyを選択

評価指標は正解率を示すaccuracyを選択

```
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
model.summary()
```



この誤差を損失関数が  
計算して最適化

# 学習モデルの作成

model.summary()は、  
作成した学習モデルを要約する

Layer (type)	Output Shape	Param #
<hr/>		
flatten_182 (Flatten)	(None, 4096)	0
dense_1075 (Dense)	(None, 512)	2097664
dropout_507 (Dropout)	(None, 512)	0
dense_1076 (Dense)	(None, 256)	131328
dropout_508 (Dropout)	(None, 256)	0
dense_1077 (Dense)	(None, 128)	32896
dropout_509 (Dropout)	(None, 128)	0
dense_1078 (Dense)	(None, 64)	8256
dense_1079 (Dense)	(None, 32)	2080
dense_1080 (Dense)	(None, 1)	33
<hr/>		
Total params: 2,272,257		
Trainable params: 2,272,257		
Non-trainable params: 0		
<hr/>		

model.compile(loss='binary\_crossentropy', optimizer='Adam', metrics=["accuracy"])  
**model.summary()**

# 学習モデルの作成

“ 5)神経回路の作成をコピーして実行してみよう

```
model = Sequential()
model.add(Flatten(input_shape=(64, 64, 1)))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adam',
metrics=["accuracy"])
model.summary()
```

## 学習と検証

```
result= model.fit(images_train, labels_train, batch_size = 64, epochs = 100, validation_split = 0.2)
```

model\_historyに学習過程を記録する。

学習用のデータを用いて、64例ずつデータを抽出して学習を行う。batch\_size = 32

全体のデータを100回使って学習を行う。epochs = 100 (1epochで学習データ全体を学習)

学習データの2割を検証用に分割して評価する。validation\_split = 0.2

## 学習と検証

```
result= model.fit(images_train, labels_train, batch_size = 64, epochs = 100, validation_split = 0.2)
```

model\_historyに学習過程を記録する。

学習用のデータを用いて、64例ずつデータを抽出して学習を行う。batch\_size = 64

全体のデータを100回使って学習を行う。epochs =100 (1epochで学習データ全体を学習)

学習データの2割を検証用に分割して評価する。validation\_split = 0.2

```
In [281]: result = model.fit(images_train, labels_train, batch_size = 64, epochs = 100, validation_split = 0.2)
Epoch 1/100
3/3 [=====] - 0s 17ms/step - loss: 0.3341 - accuracy: 0.8541 - val_loss: 0.3681 - val_accuracy: 0.8936
Epoch 2/100
3/3 [=====] - 0s 10ms/step - loss: 0.2973 - accuracy: 0.8919 - val_loss: 0.4421 - val_accuracy: 0.7234
Epoch 3/100
3/3 [=====] - 0s 10ms/step - loss: 0.3467 - accuracy: 0.8649 - val_loss: 0.3417 - val_accuracy: 0.8723
Epoch 4/100
3/3 [=====] - 0s 10ms/step - loss: 0.3116 - accuracy: 0.8919 - val_loss: 0.3454 - val_accuracy: 0.8936
[... 5/100]
```

Epoch 1/100 → epochs=100なので100/100まで学習

3/3 → validation\_split = 0.2なので、学習に用いるデータ数は $232 \times 0.8 = 185.6$

batch\_size = 64なので  $64 \times 2 < 185.6 < 64 \times 3$  で1epoch

## 学習と検証

```
Epoch 1/100
3/3 [=====] - 0s 30ms/step - loss: 0.7406 - accuracy: 0.4703 - val_loss: 0.8413 - val_accuracy: 0.4468
Epoch 2/100
3/3 [=====] - 0s 10ms/step - loss: 0.7753 - accuracy: 0.5405 - val_loss: 0.7463 - val_accuracy: 0.5532
Epoch 3/100
3/3 [=====] - 0s 9ms/step - loss: 0.8044 - accuracy: 0.4703 - val_loss: 0.6926 - val_accuracy: 0.5532
Epoch 4/100
3/3 [=====] - 0s 9ms/step - loss: 0.7331 - accuracy: 0.5405 - val_loss: 0.6425 - val_accuracy: 0.5532
Epoch 5/100
:
:
:
Epoch 98/100
3/3 [=====] - 0s 9ms/step - loss: 0.4393 - accuracy: 0.8108 - val_loss: 0.5829 - val_accuracy: 0.7447
Epoch 99/100
3/3 [=====] - 0s 9ms/step - loss: 0.3951 - accuracy: 0.8108 - val_loss: 0.4356 - val_accuracy: 0.8298
Epoch 100/100
3/3 [=====] - 0s 10ms/step - loss: 0.4478 - accuracy: 0.7946 - val_loss: 0.4315 - val_accuracy: 0.7872
```

実行するとコンソール画面に学習過程が出力される

loss: 学習用データの誤差、0に近いほどよい

accuracy: 学習用データの正解率、1に近いほどよい

val\_loss: 検証用データの誤差、0に近いほどよい

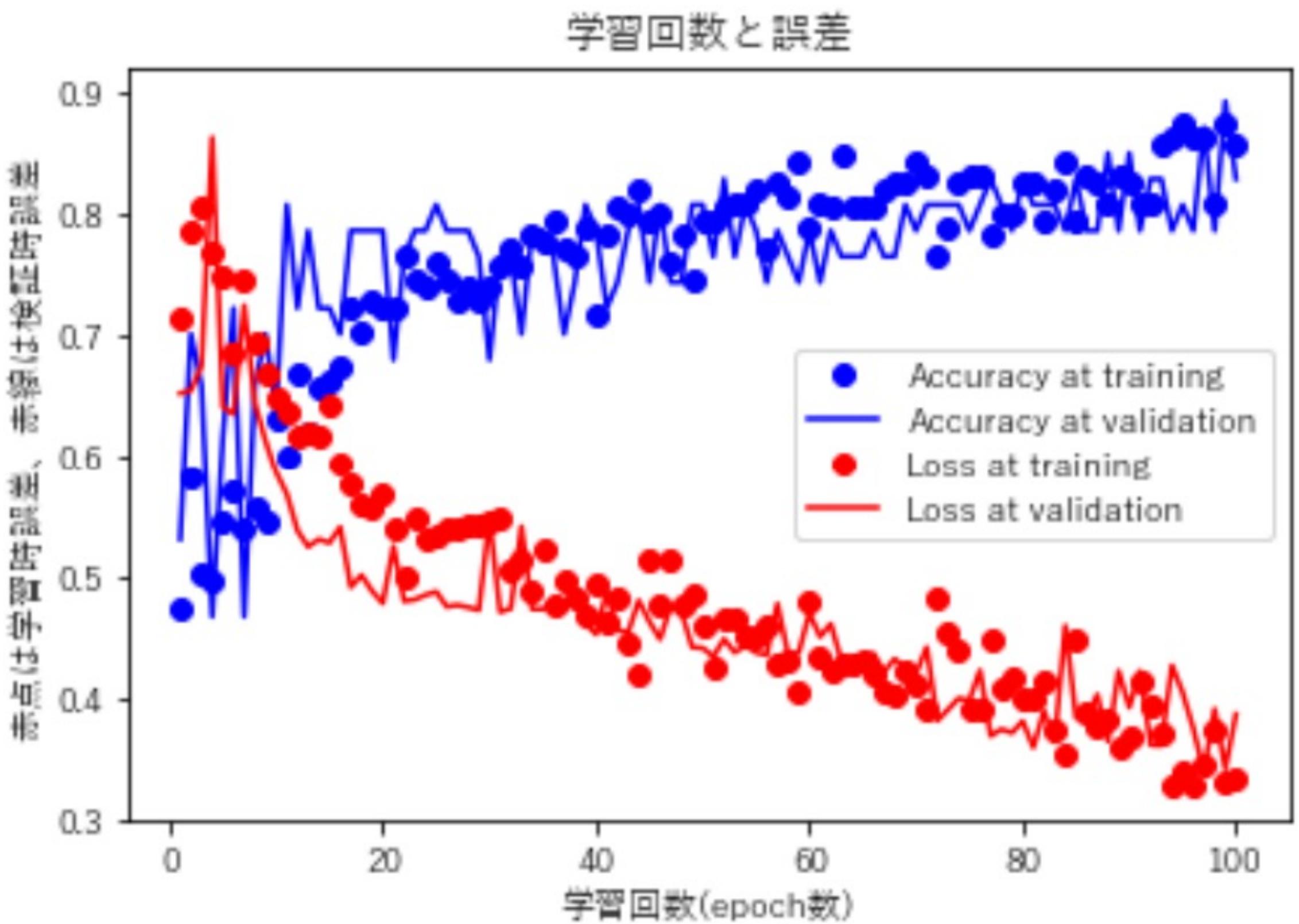
val\_accuracy: 検証用データの正解率、1に近いほどよい

# 結果の可視化

“7) 学習過程をグラフ表示する”を実行

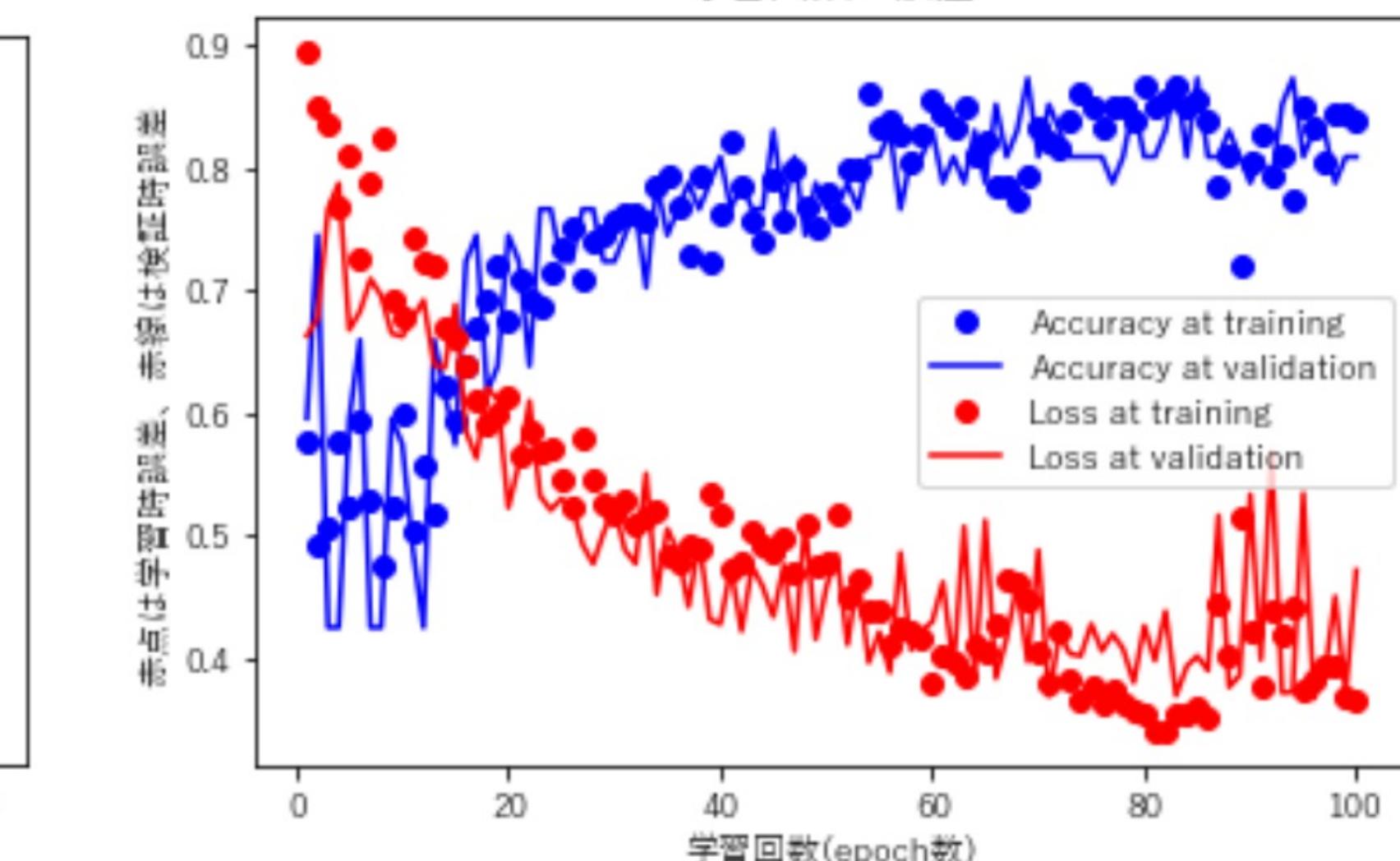
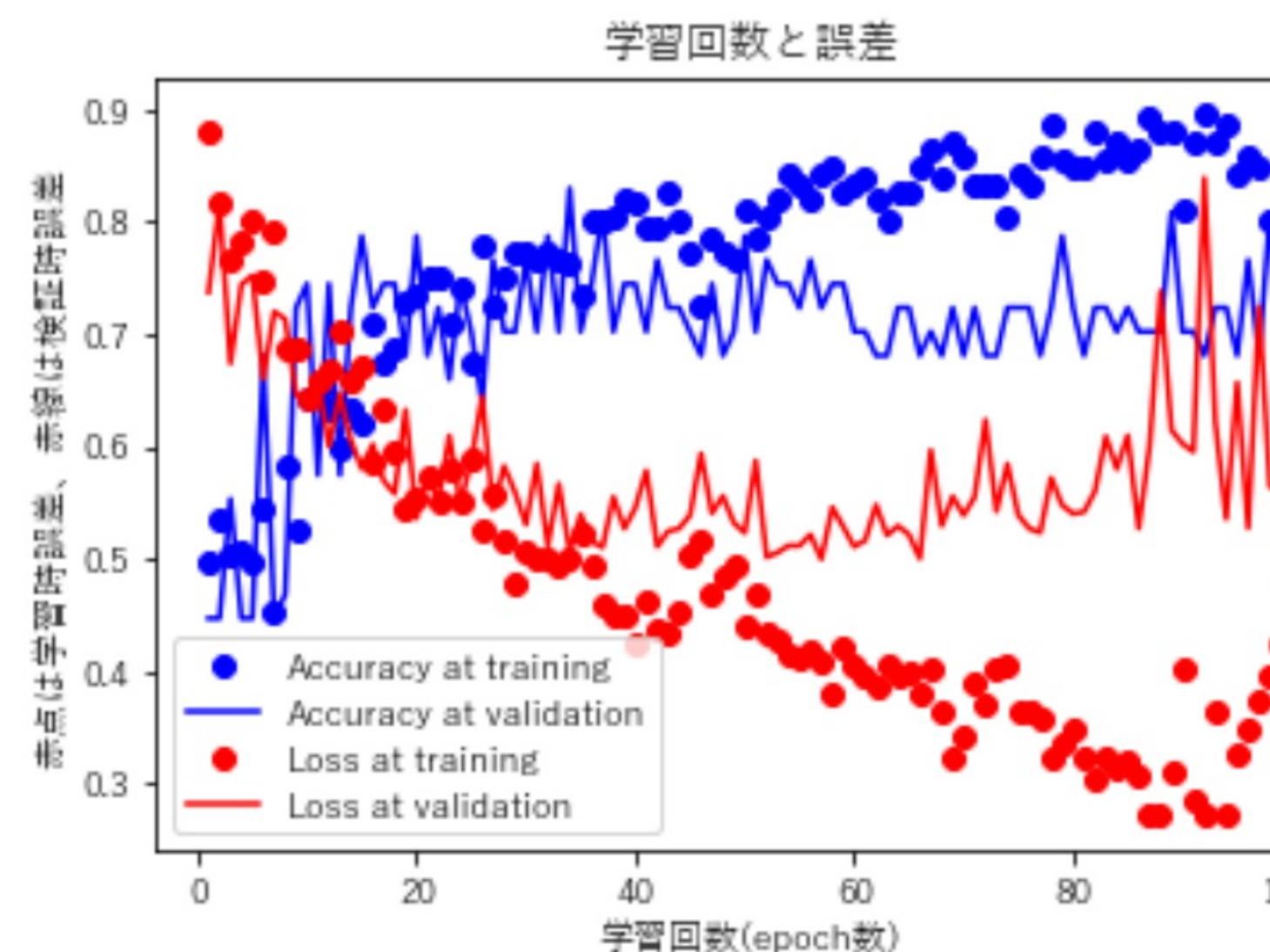
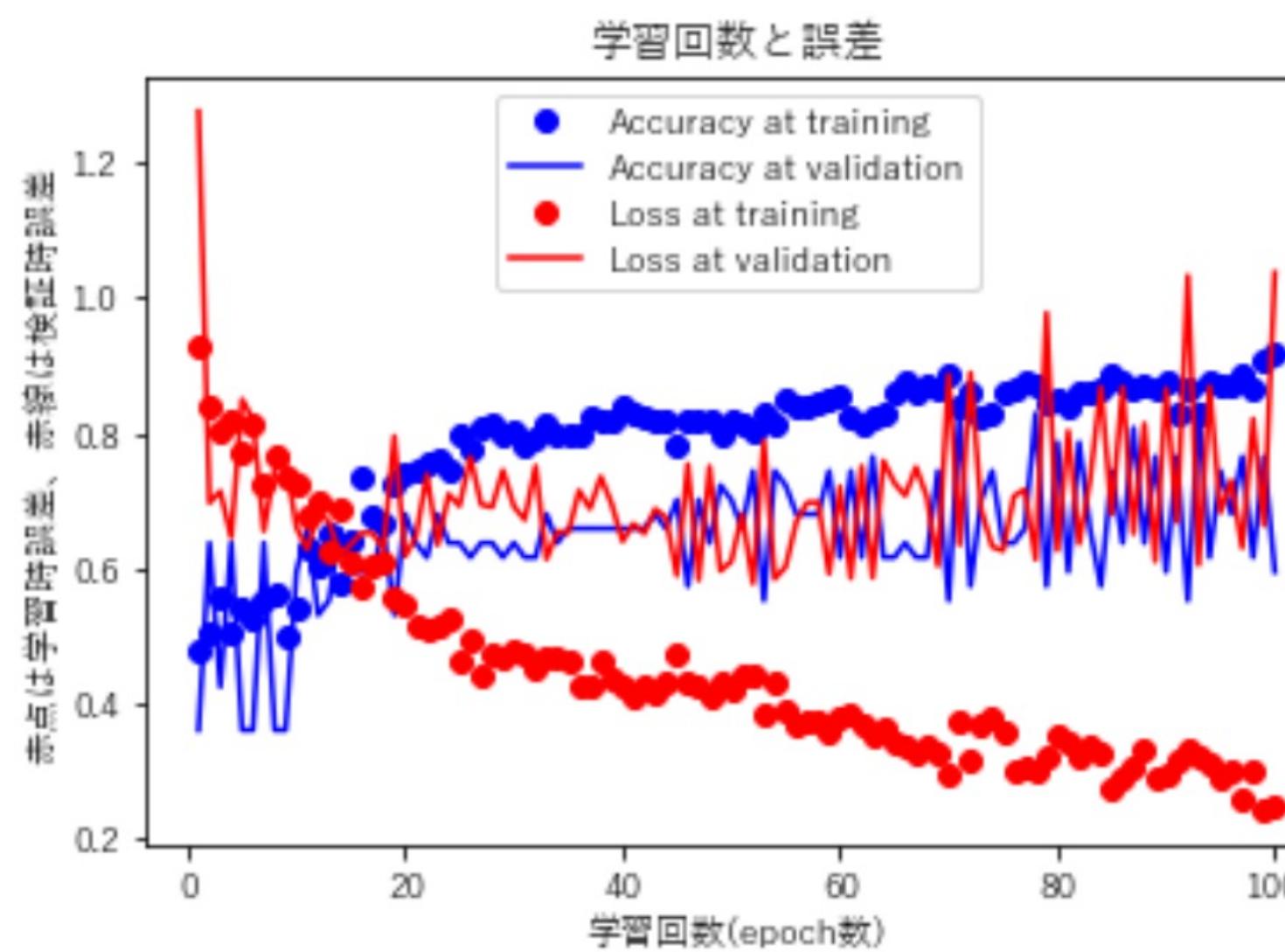
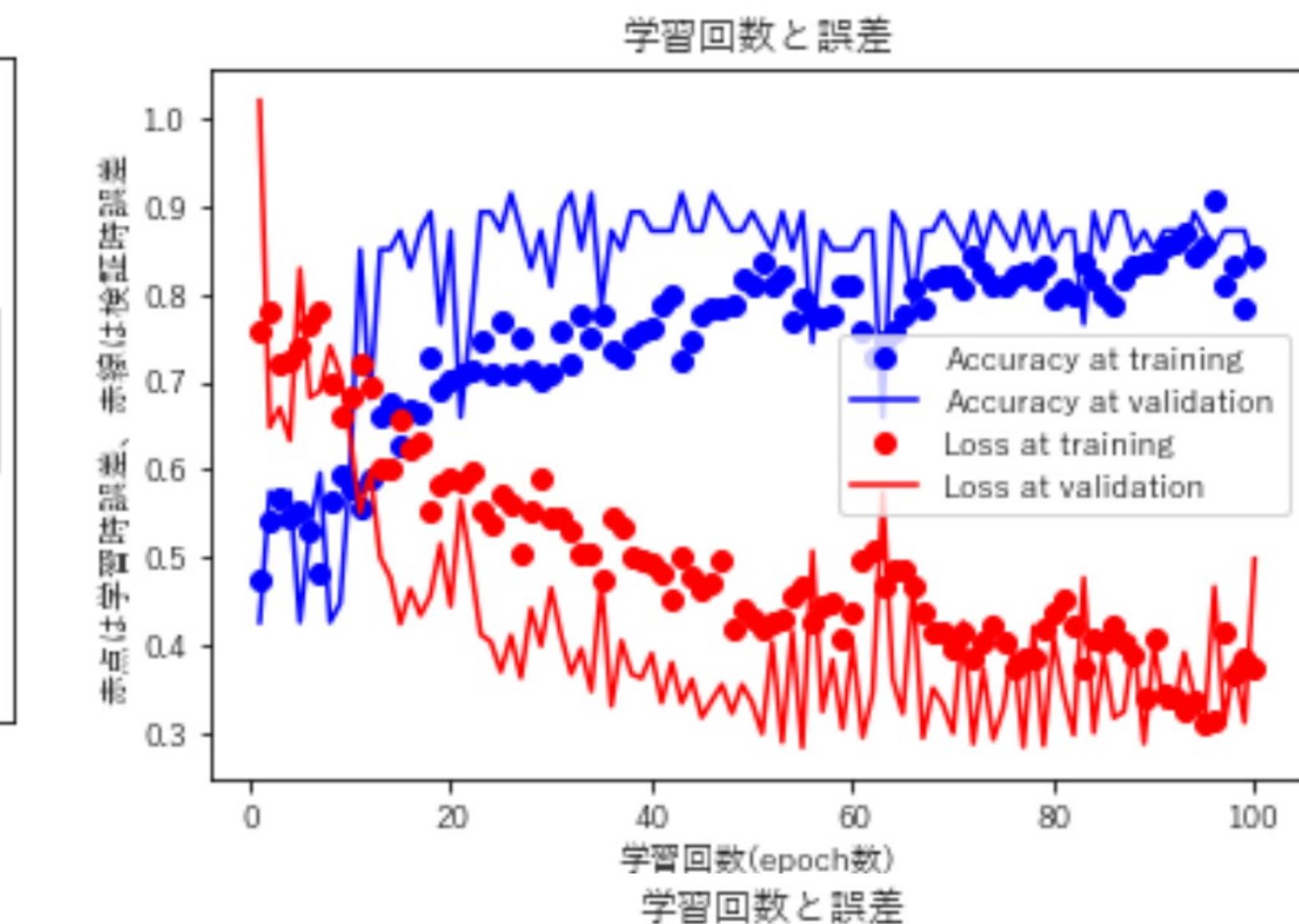
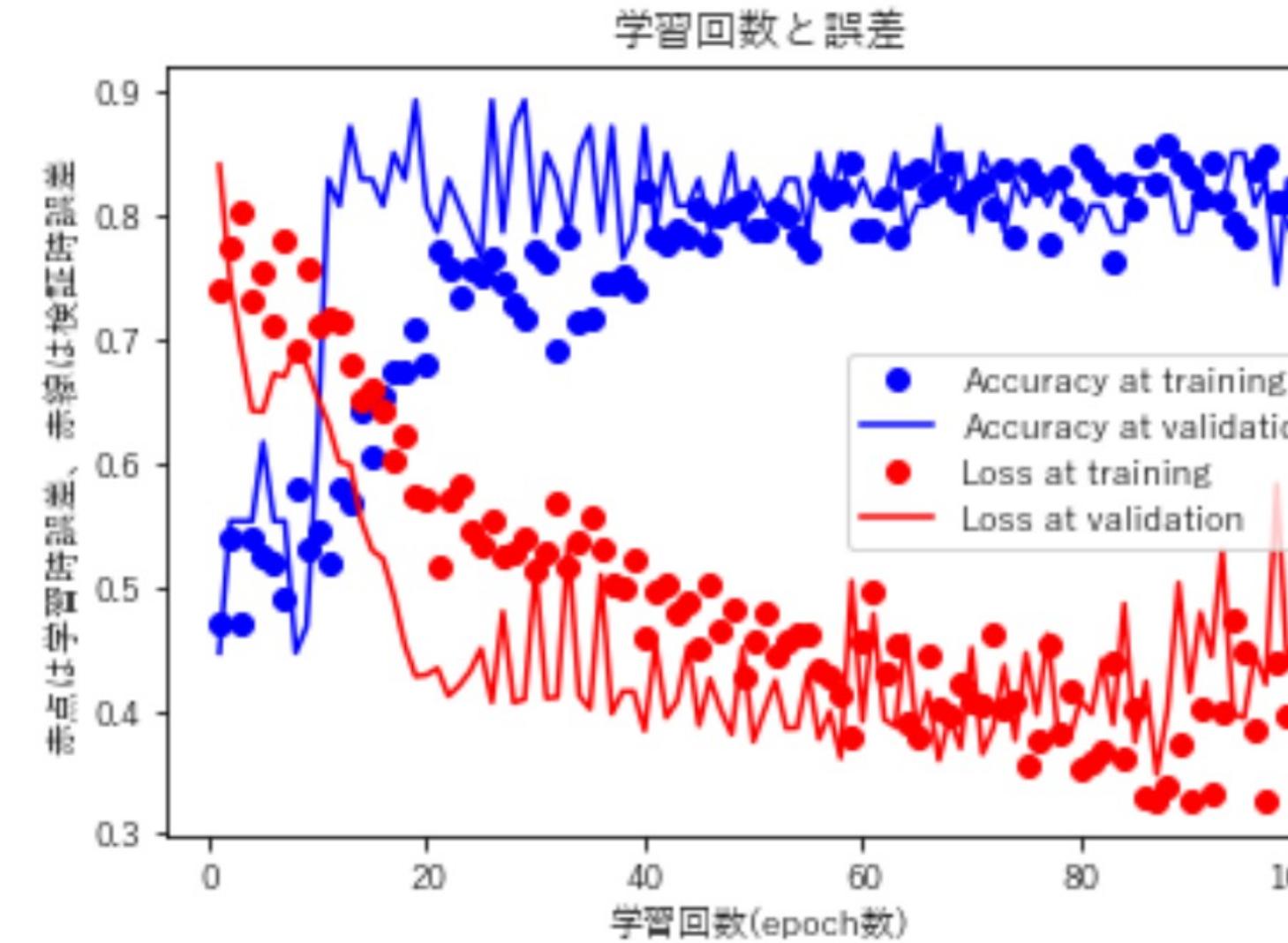
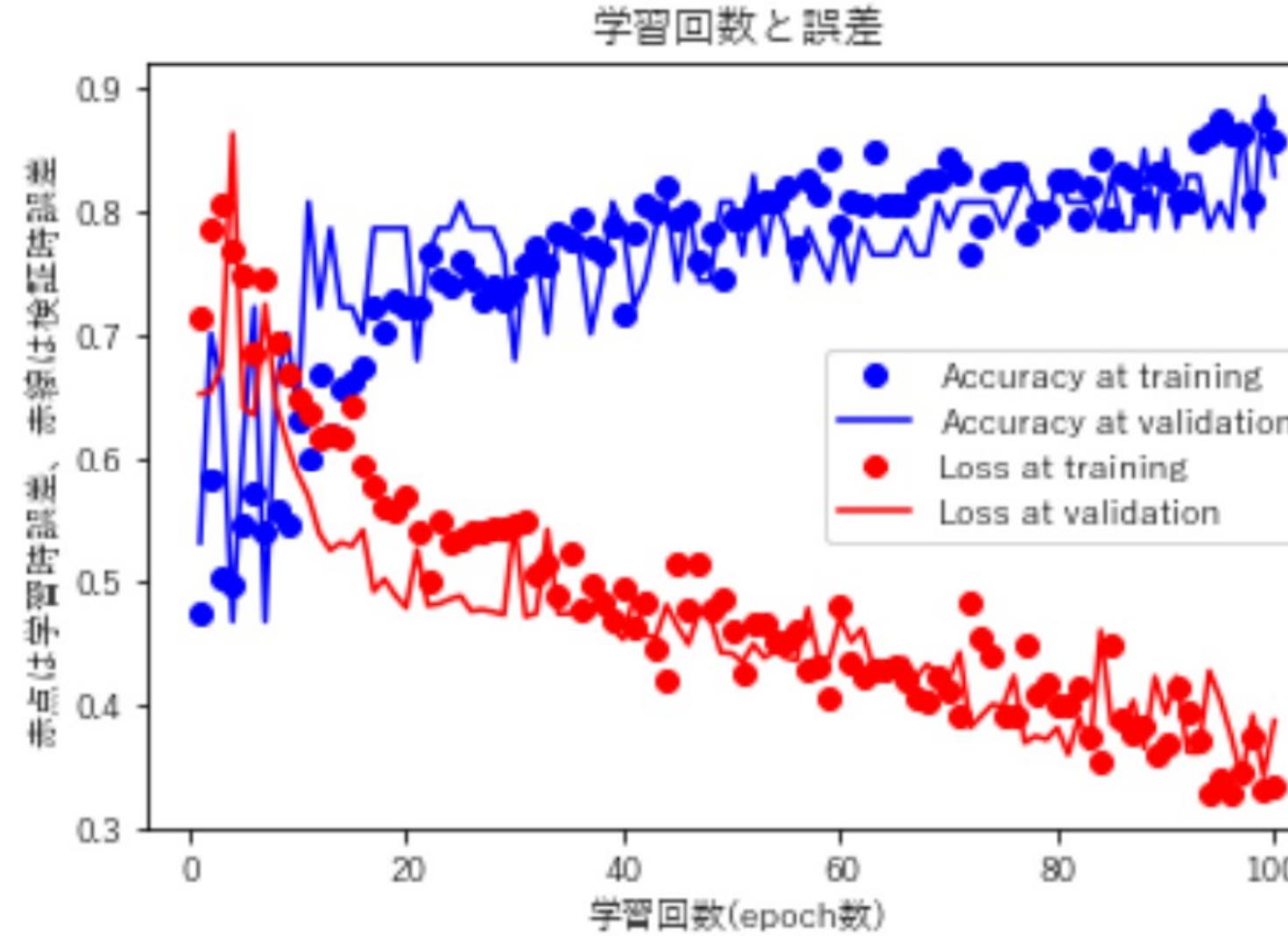
青が正解率、赤が誤差  
点が学習データ、線が検証用データ

```
# Loss(正解との誤差)をloss_valuesに入れる
loss = result.history['loss']
val_loss = result.history['val_loss']
# 正確度をaccに入れる
acc = result.history['accuracy']
val_acc = result.history['val_accuracy']
# 1からepoch数までのリストを作る
epochlist = range(1, len(loss) + 1)
# 正確度のグラフを作る
# 'bo'は青い線
plt.plot(epochlist, acc, 'bo', label='Accuracy at training')
plt.plot(epochlist, val_acc, 'b', label='Accuracy at validation')
# Loss(正解との誤差)のグラフを作る
# 'ro'は赤い点 https://matplotlib.org/api/\_as\_gen/matplotlib.pyplot.plot.html
plt.plot(epochlist, loss, 'ro', label='Loss at training')
plt.plot(epochlist, val_loss, 'r', label='Loss at validation')
# タイトル
plt.title('学習回数と誤差')
plt.ylabel('点は学習、線は検証、赤は損失、青は正解率')
plt.xlabel('学習回数(epoch数)')
plt.legend()
plt.show()
```



学習する度に正解率が上がって誤差が下がっていることが分かる

学習は毎回ランダムなパラメータを更新するので毎回結果が変わります。  
(今回は学習するデータも少ないためあまり安定しません)

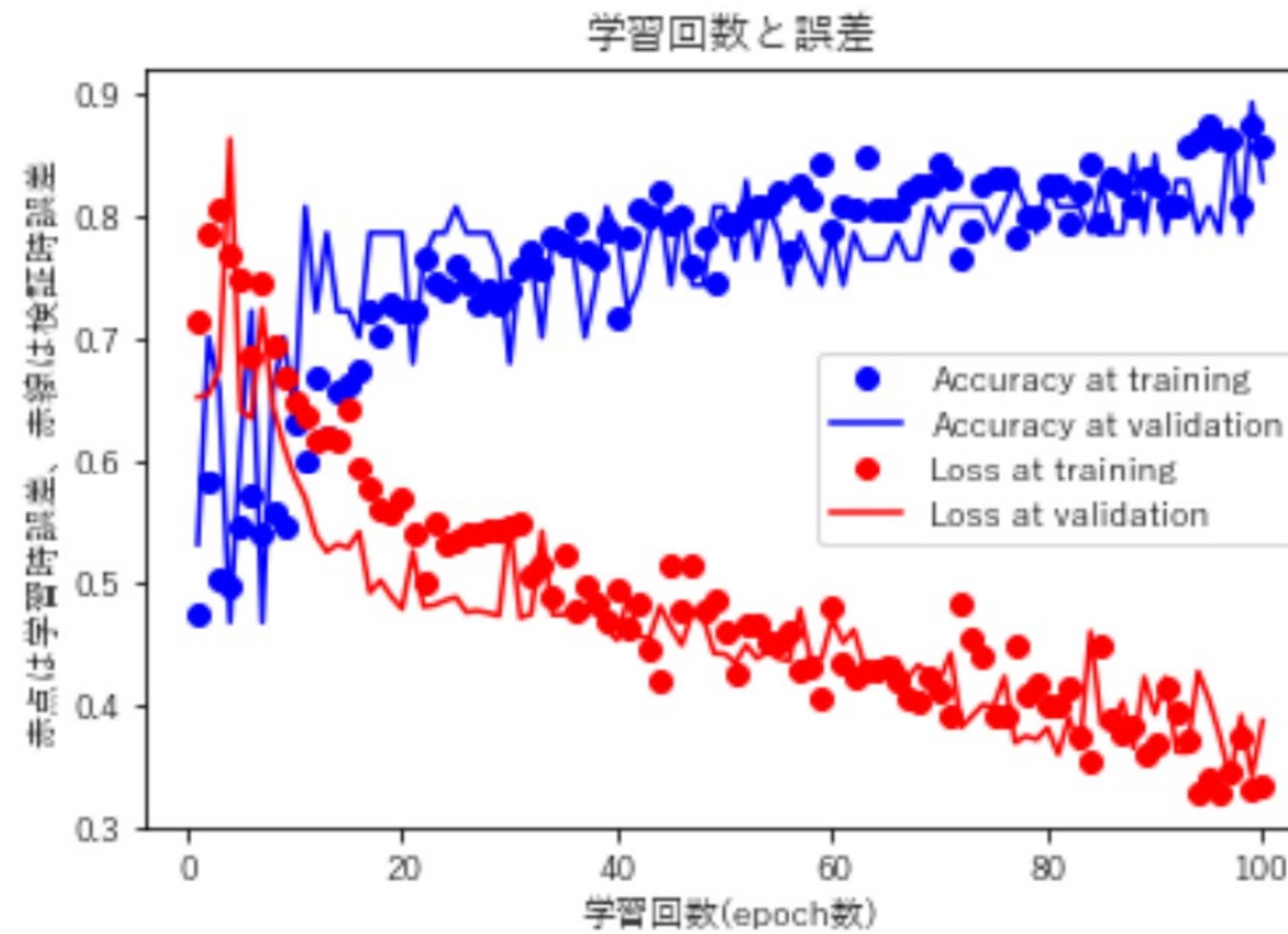


## #5と#6を消して#8に置き換えてみよう

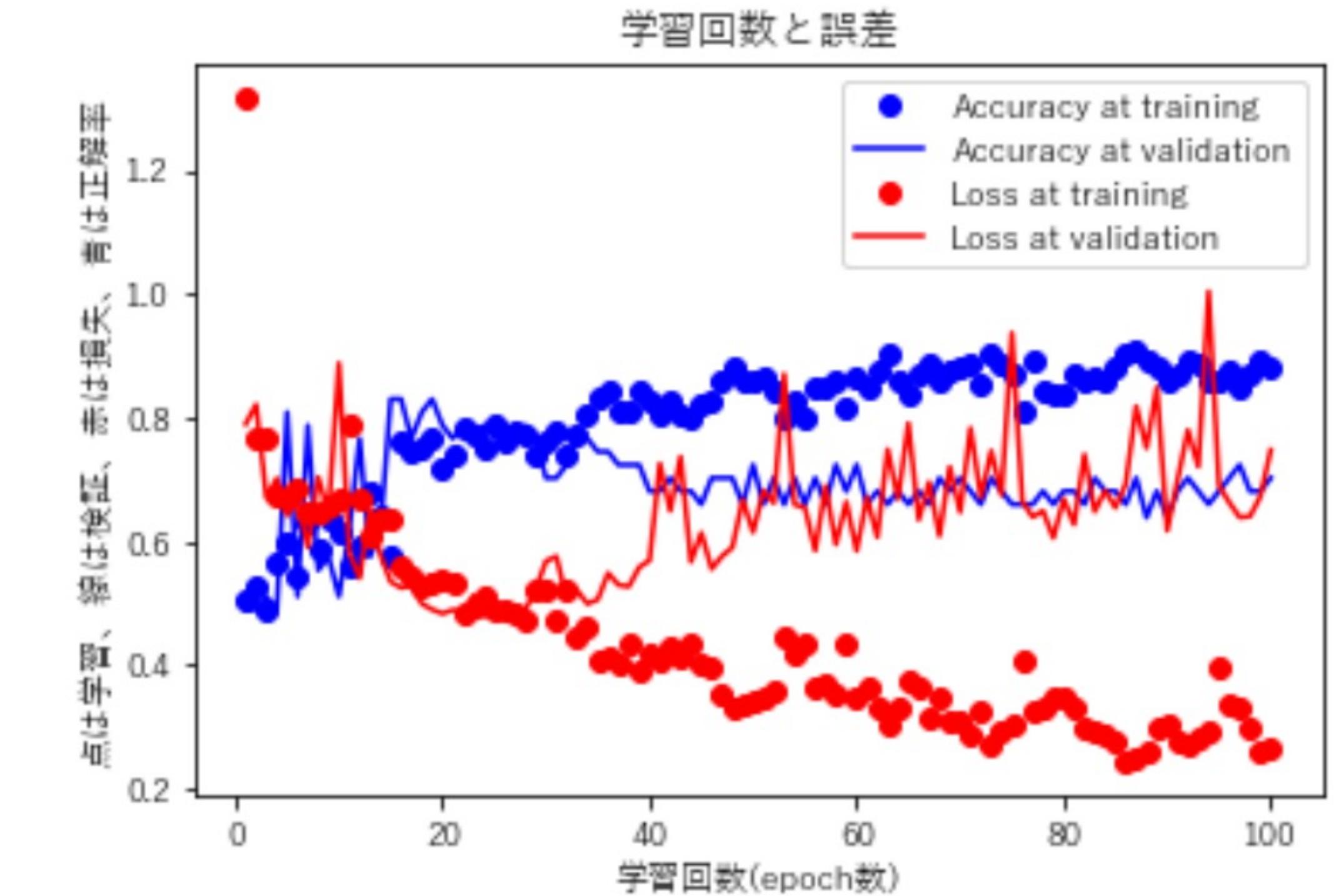
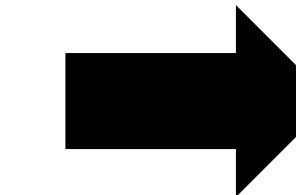
```
model = Sequential()
model.add(Flatten(input_shape=(64, 64, 1)))
model.add(Dense(512, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
# model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
# model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
model.summary()
```

# Dropoutを行わずに実行してみる

学習データ(点)が結果が良いが、検証用データ(線)が悪い → 過学習

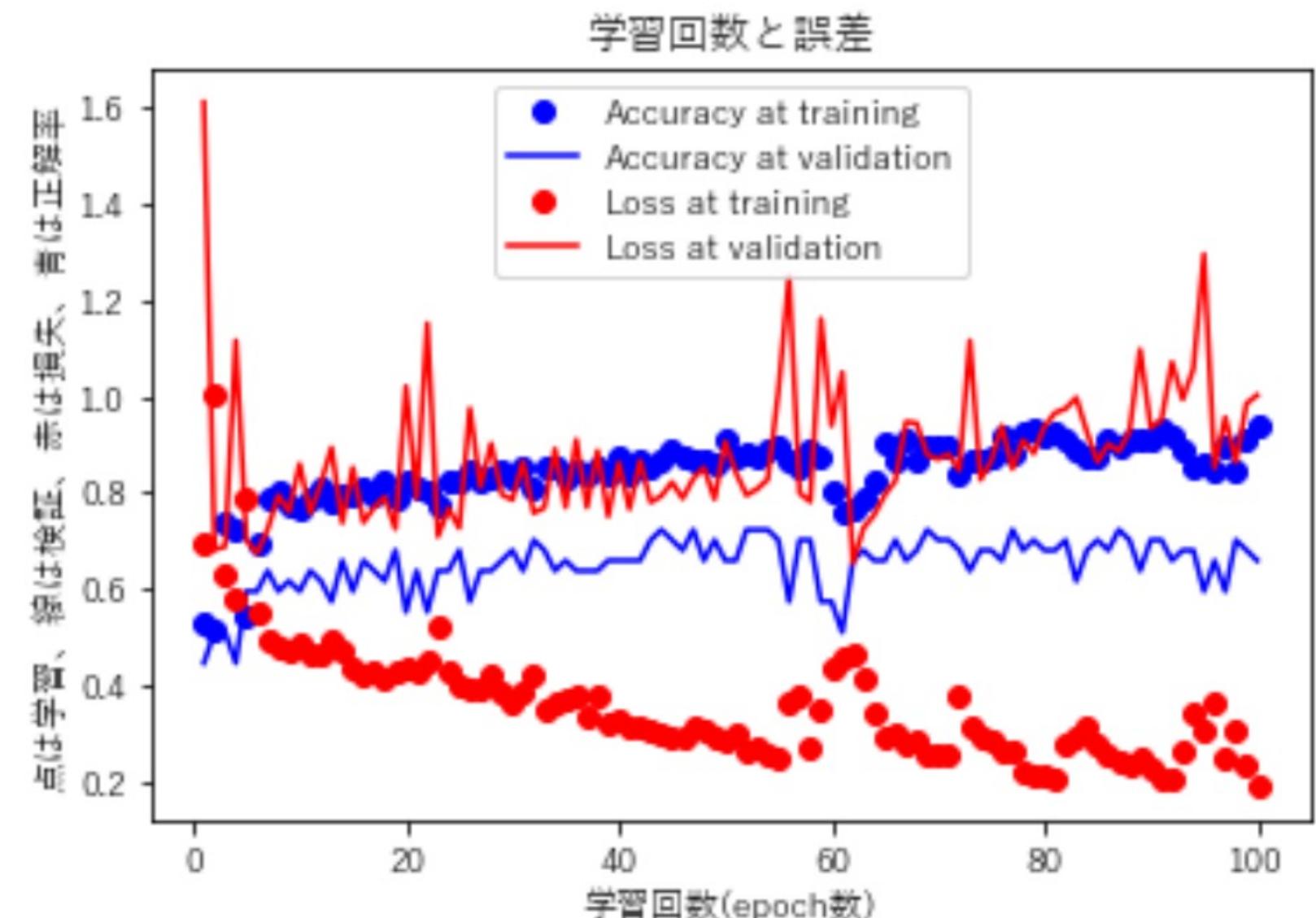
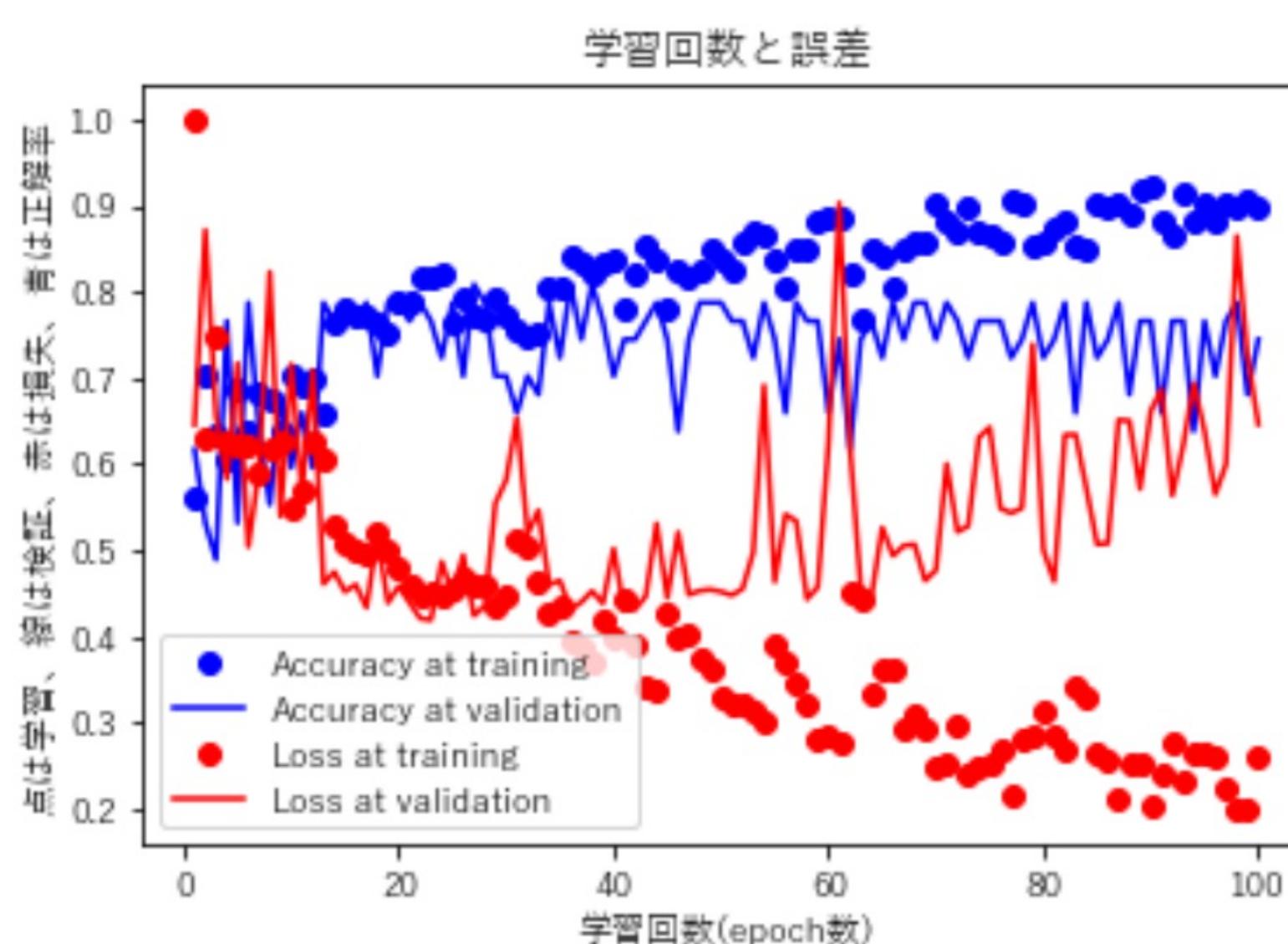
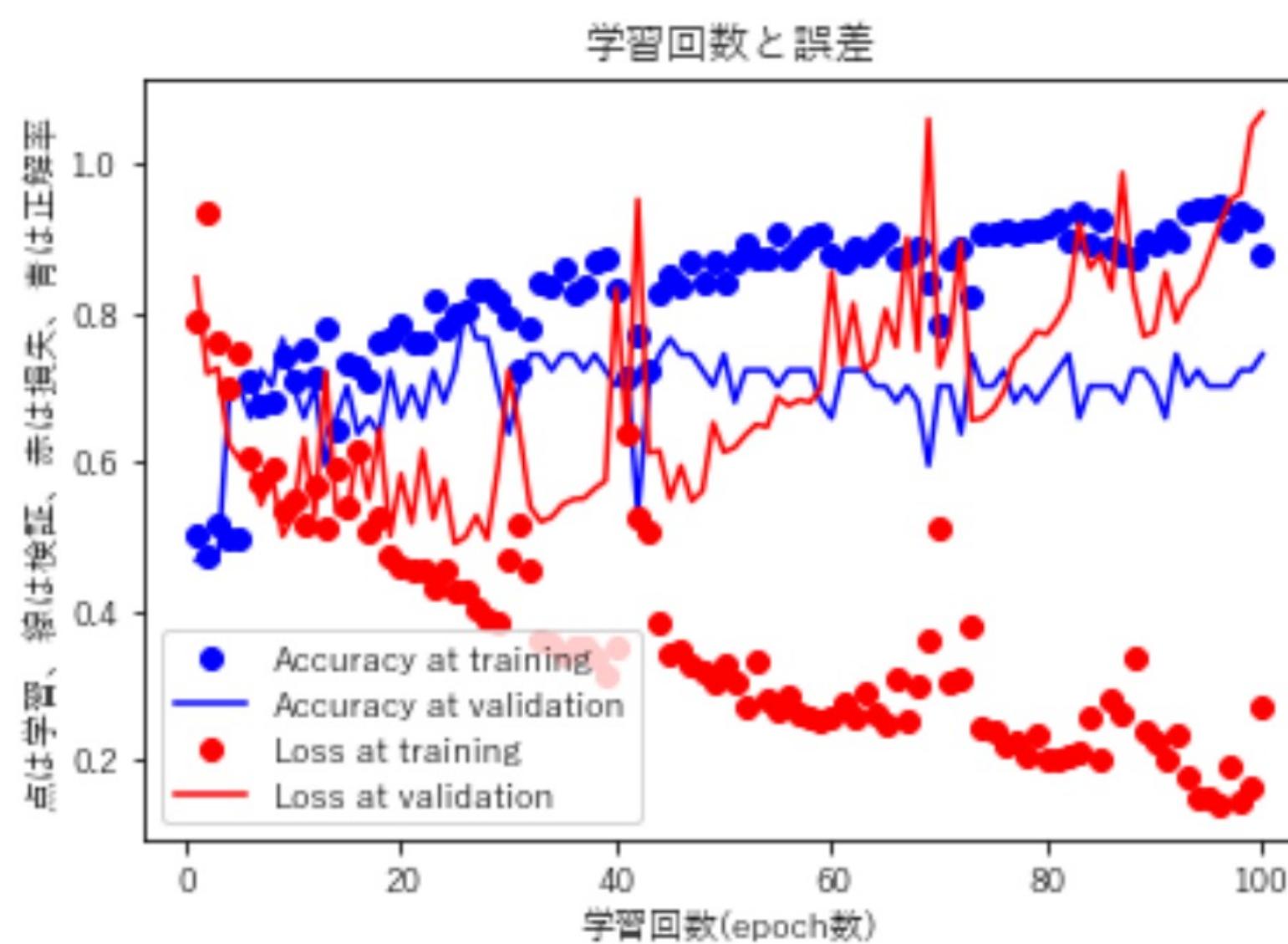
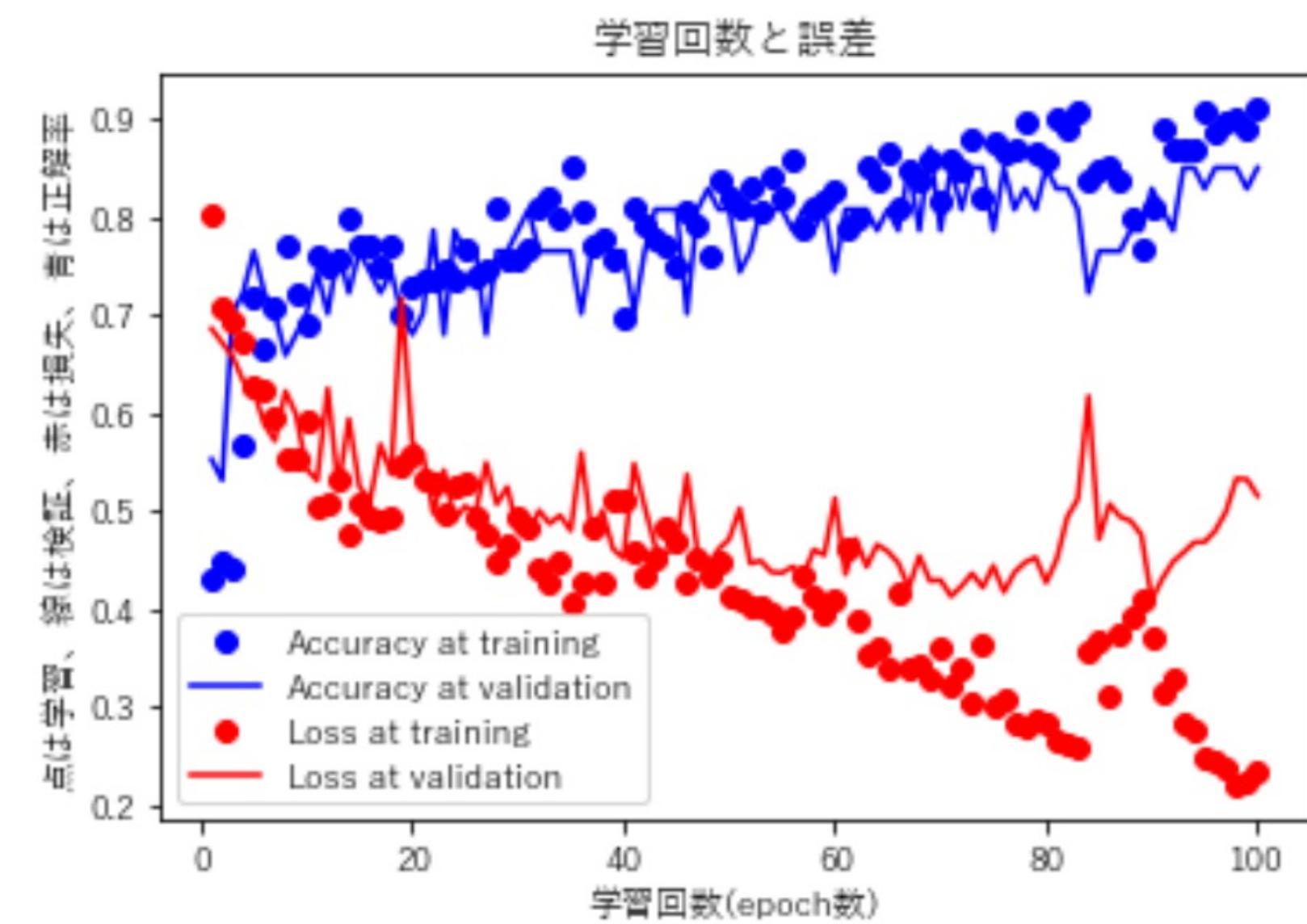
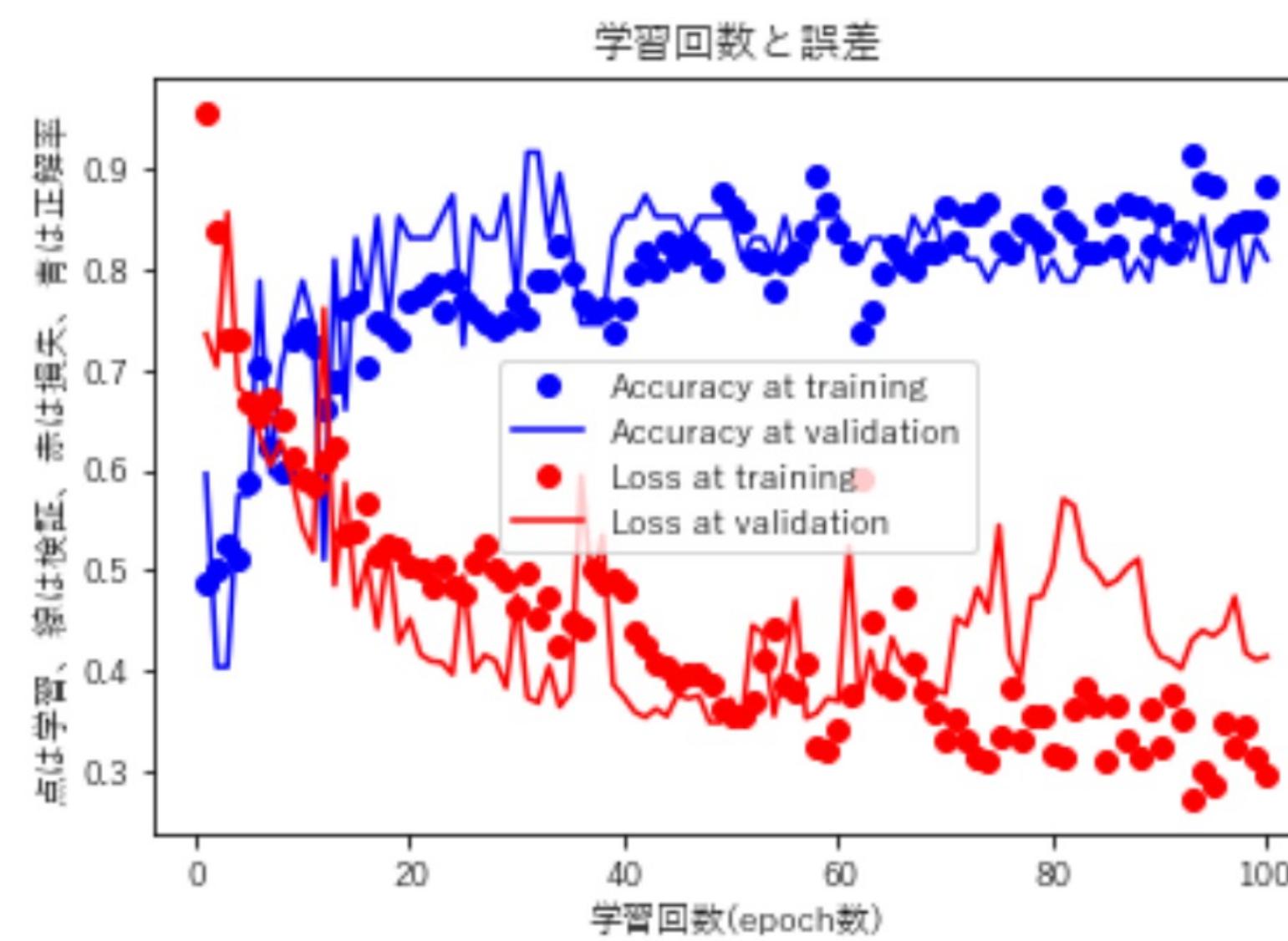
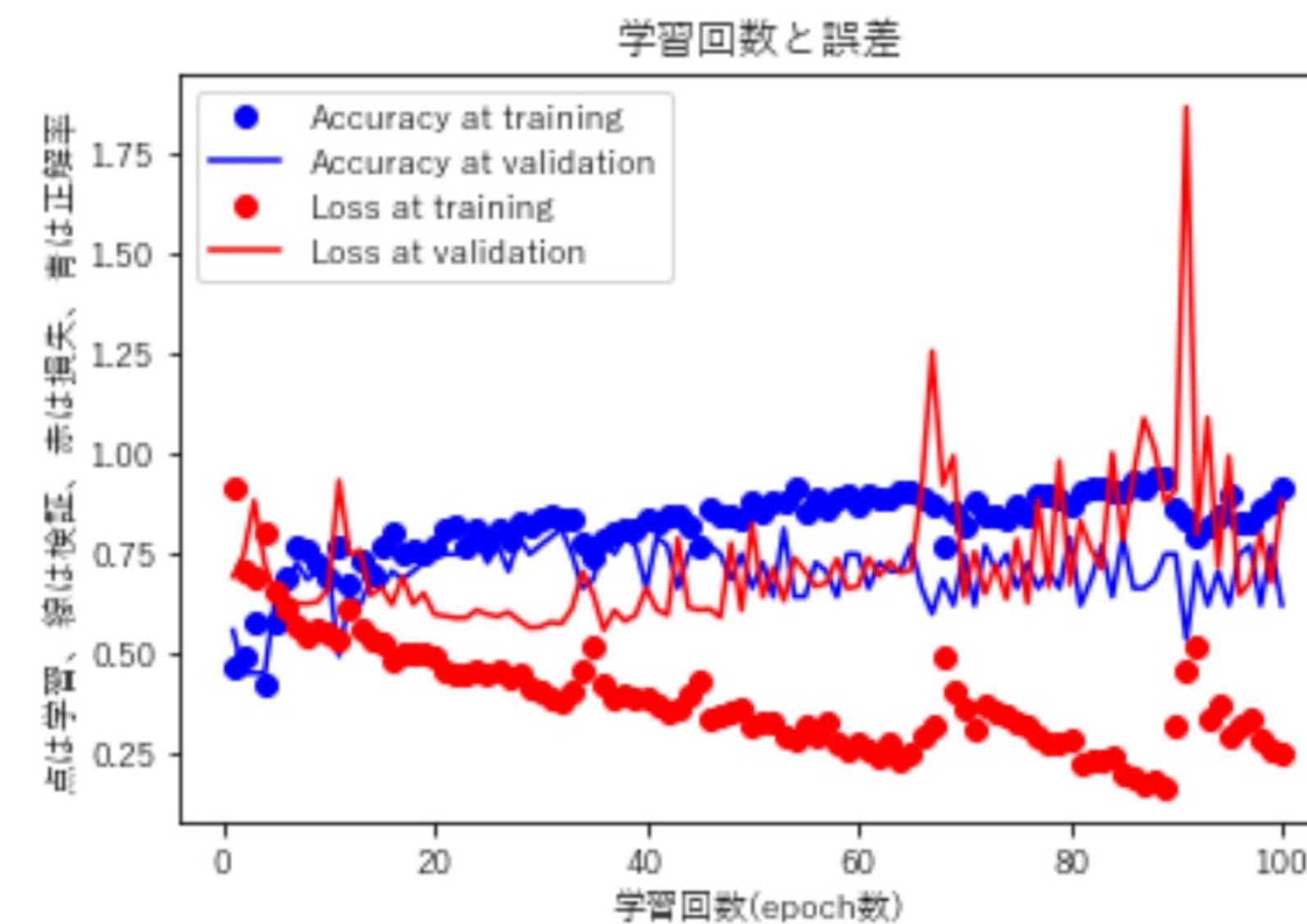


Dropout有り



Dropout無し

- ・学習時に検証用を分けることで過学習が起きてないか確認することができる
- ・Dropoutは過学習の抑制に役立つ



# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64))/255
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64))/255
check = np.zeros((2,64,64,1))
check[0] = img1
check[1] = img2
myprobs = model.predict(check)
print(myprobs)
print("%.5f" %myprobs[0])
print("%.5f" %myprobs[1])
```



covid1.jpeg



NORMAL.jpeg

# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64)))/255
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64)))/255
check = np.zeros((2,64,64,1))
check[0] = img1
check[1] = img2
myprobs = model.predict(check)
print(myprobs)
print("%.5f" % myprobs[0])
print("%.5f" % myprobs[1])
```

画像を白黒で読み込んでnumpy配列に変換したものを255で割る

# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64)))/255
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64)))/255
check = np.zeros((2,64,64,1))
check[0] = img1
check[1] = img2
myprobs = model.predict(check)
print(myprobs)
print("%.5f" % myprobs[0])
print("%.5f" % myprobs[1])
```

画像を白黒で読み込んでnumpy配列に変換したものを255で割る

(2, 64, 64, 1)の4次元配列の行列を作って、画像データをcheck[0]とcheck[1]に代入

# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
check = np.zeros((2,64,64,1))  
check[0] = img1  
check[1] = img2  
myprobs = model.predict(check)  
print(myprobs)  
print("%.5f" %myprobs[0])  
print("%.5f" %myprobs[1])
```

画像を白黒で読み込んでnumpy配列に変換したものを255で割る

(2, 64, 64, 1)の4次元配列の行列を作って、画像データをcheck[0]とcheck[1]に代入

model.predict()で正解率の予測

# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
check = np.zeros((2,64,64,1))  
check[0] = img1  
check[1] = img2  
myprobs = model.predict(check)  
print(myprobs)  
print("%.5f" %myprobs[0])  
print("%.5f" %myprobs[1])
```

画像を白黒で読み込んでnumpy配列に変換したものを255で割る

(2, 64, 64, 1)の4次元配列の行列を作って、画像データをcheck[0]とcheck[1]に代入

model.predict()で正解率の予測

そのまま出力すると指数表記で読みづらい

```
print(myprobs)  
[[8.2333648e-01]  
 [3.2362335e-05]]
```

# 新たな画像で分類を試してみよう

2枚の画像で検証してみよう

```
img1 = img_to_array(load_img('./check_images/covid1.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
img2 = img_to_array(load_img('./check_images/NORMAL.jpeg', color_mode='grayscale', target_size=(64,64)))/255  
check = np.zeros((2,64,64,1))  
check[0] = img1  
check[1] = img2  
myprobs = model.predict(check)  
print(myprobs)  
print("%.5f" %myprobs[0])  
print("%.5f" %myprobs[1])
```

「%.5f」%変数」で小数第5位まで出力する

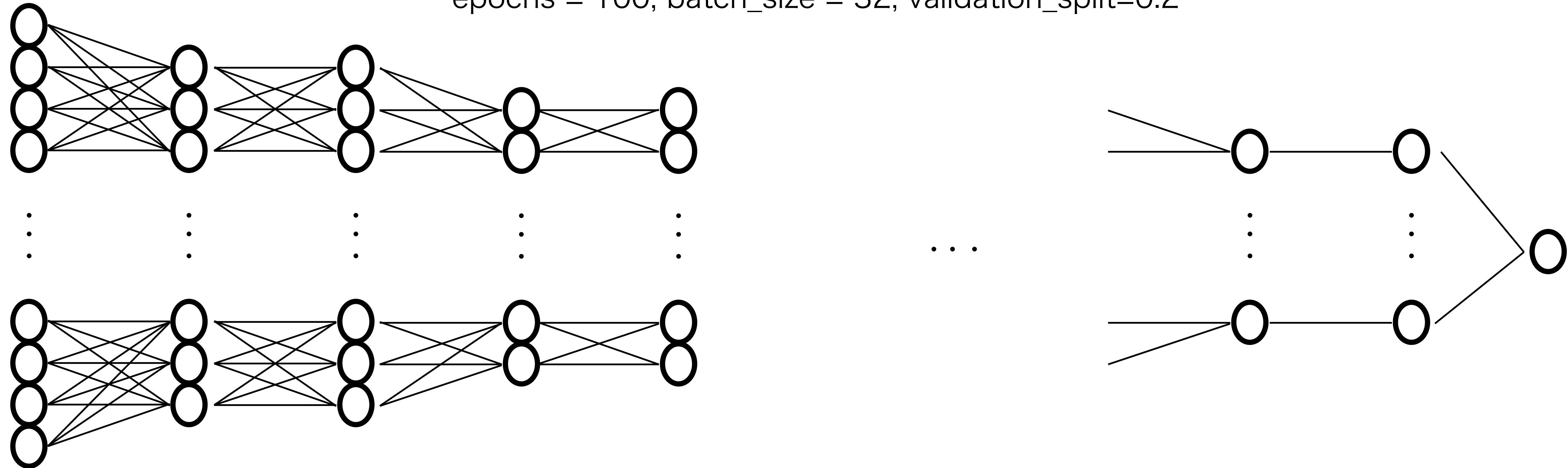
```
print("%.5f" %myprobs[0])  
0.82334
```

```
print("%.5f" %myprobs[1])  
0.00003
```

covid1.jpegは0.82334の確率でcovid19、  
NORMAL.jpegは0.00003の確率でcovid19  
と分類された

# 次の学習モデルを作成して学習結果を図示

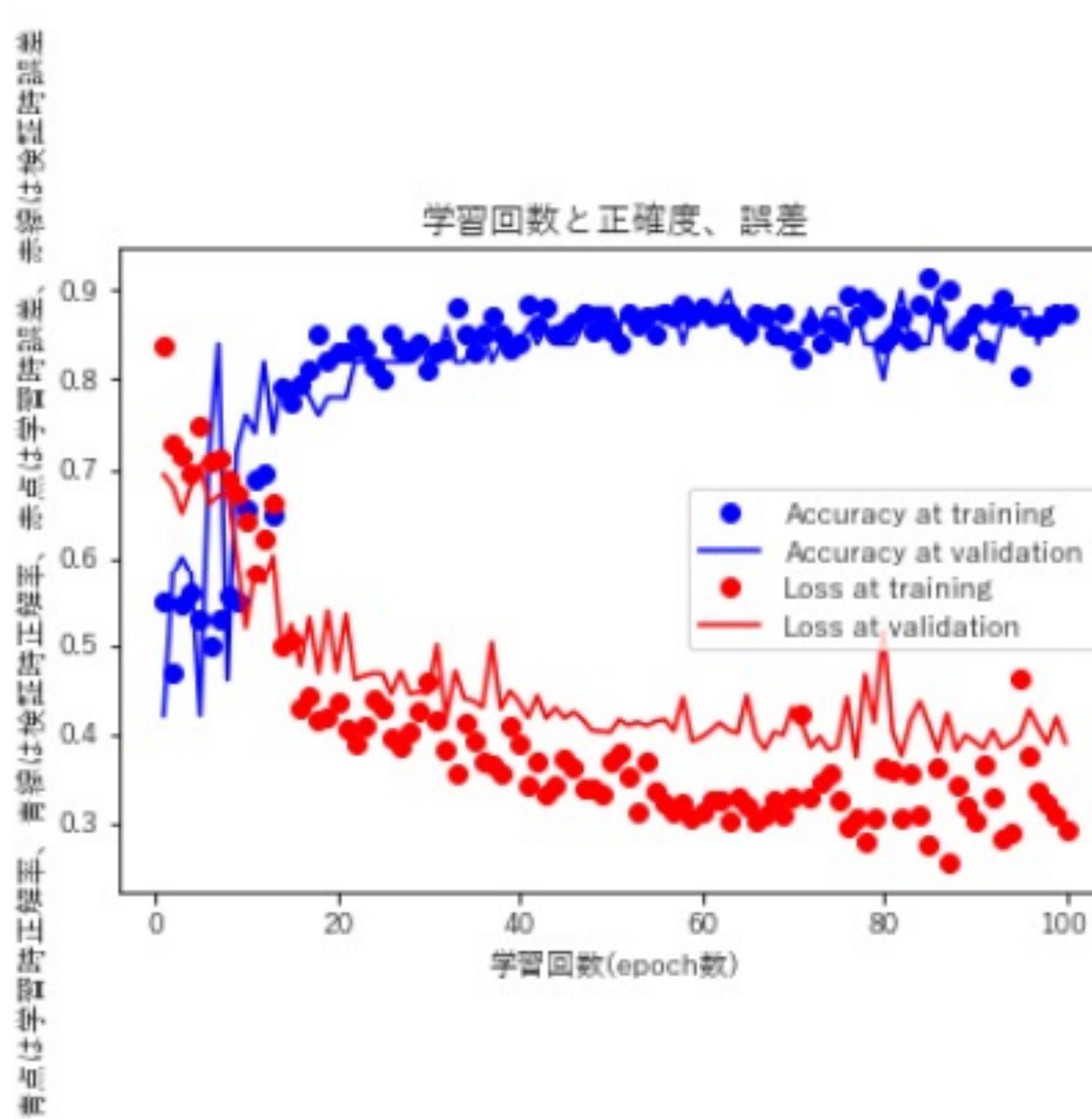
epochs = 100, batch\_size = 32, validation\_split=0.2



入力層	1層目	2層目	3層目	4層目	5層目	6層目	7層目	8層目	9層目
4096個	1024個	1024個	512個	512個	256個	128個	64個	32個	16個
Flatten	Dense	Dropout (0.5)	Dense	Dropout (0.5)	Dense	Dense	Dense	Dense	Dropout

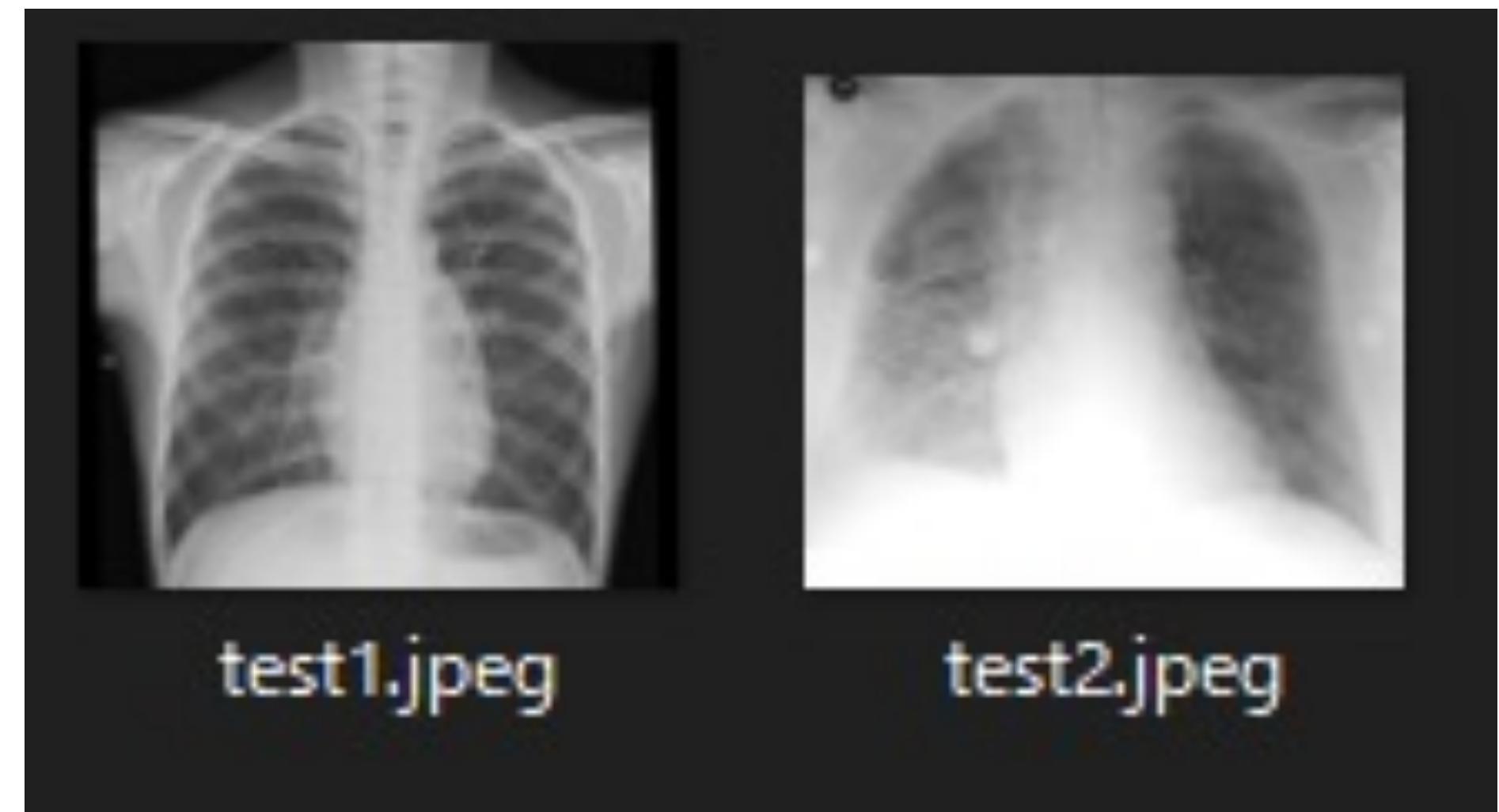
# 学習モデルの作成

```
model = Sequential()
model.add(Flatten(input_shape=(64, 64, 1)))
model.add(Dense(units=1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=256, activation='relu'))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
model.summary()
model_history = model.fit(images_train, labels_train, batch_size = 32, epochs = 100,
validation_split = 0.2)
```



# 課題の考え方

check\_imagesのtest1.jpegとtest2.jpegの  
2枚の画像がcovid19である確率を求めよう



学習回数を順に上げてもらう  $50 \rightarrow 100 \rightarrow 200 \rightarrow 500$   
validation\_splitを変えてもらう  $0.2 \rightarrow 0.3 \rightarrow 0.5$   
層を増やしてもらう  
層の中のニューロンの数を変えてもらう  
dropoutを0にしてもらう

変化を見てもらう？

# 自分たちで実践してみよう

2層目から4層目のニューロンを1つ増やして、  
学習の回数を500回にして実践してみよう

