simple convolution network

image to column

In [3]:
```
import sys, os
sys.path.append(os.pardir)
import pickle
import numpy as np
from collections import OrderedDict
from common import layers
from common import optimizer
from data.mnist import load_mnist
import matplotlib.pyplot as plt

# 画像データを２次元配列に変換
'''
input_data: 入力値
filter_h: フィルターの高さ
filter_w: フィルターの横幅
stride: ストライド
pad: パディング
'''
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h)//stride + 1 #1は初期位置分
    out_w = (W + 2 * pad - filter_w)//stride + 1

    #input_dataの配列に対して、numberとchannelはpaddingしない。height方向とwidth方向にint(pad)個のpaddingを行う。
    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3) # (N, C, filter_h, filter_w, out_h, out_w) -> (N, filter_w, out_h, out_w, C, filter_h)

    col = col.reshape(N * out_h * out_w, -1)
    return col
```

[try] im2colの処理を確認しよう

・関数内でtransposeの処理をしている行をコメントアウトして下のコードを実行してみよう
・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

In [5]:
```
# im2colの処理確認
input_data = np.random.rand(2, 1, 4, 4)*100//1 # number, channel, height, widthを表す
print('========== input_data ==========¥n', input_data)
print('==============================')
filter_h = 2
filter_w = 2
stride = 1
pad = 0
col = im2col(input_data, filter_h=filter_h, filter_w=filter_w, stride=stride, pad=pad)
print('============= col ==============¥n', col)
print('==============================')
```

```
========== input_data ===========
 [[[[81. 37. 60. 61.]
    [ 7. 87. 81. 37.]
    [49. 37. 35. 95.]
    [18.  4. 96. 78.]]]


  [[[ 6. 13. 51. 75.]
    [77.  6. 83. 50.]
    [60.  8. 67. 21.]
    [ 1. 99. 96. 39.]]]]
==============================
============= col ==============
 [[81. 37.  7. 87.]
 [37. 60. 87. 81.]
 [60. 61. 81. 37.]
 [ 7. 87. 49. 37.]
 [87. 81. 37. 35.]
 [81. 37. 35. 95.]
 [49. 37. 18.  4.]
 [37. 35.  4. 96.]
 [35. 95. 96. 78.]
 [ 6. 13. 77.  6.]
 [13. 51.  6. 83.]
 [51. 75. 83. 50.]
 [77.  6. 60.  8.]
 [ 6. 83.  8. 67.]
 [83. 50. 67. 21.]
 [60.  8.  1. 99.]
 [ 8. 67. 99. 96.]
 [67. 21. 96. 39.]]
==============================
```

column to image

In [6]:
```
# ２次元配列を画像データに変換
def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    N, C, H, W = input_shape
    # 切り捨て除算
    out_h = (H + 2 * pad - filter_h)//stride + 1 #畳み込み後の高さ
    out_w = (W + 2 * pad - filter_w)//stride + 1 #畳見込み後の幅
    col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1, 2)
    # (N, filter_h, filter_w, out_h, out_w, C)

    img = np.zeros((N, C, H + 2 * pad + stride - 1, W + 2 * pad + stride - 1))
    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]

    return img[:, :, pad:H + pad, pad:W + pad]
```

col2imの処理を確認しよう

・im2colの確認で出力したcolをimageに変換して確認しよう

In [7]:
```
print("col¥n",col,"¥n")
image = col2im(col,input_data.shape,3,3,1,0)
print("image",image,"¥n")
```

```
col
 [[81. 37.  7. 87.]
 [37. 60. 87. 81.]
 [60. 61. 81. 37.]
 [ 7. 87. 49. 37.]
 [87. 81. 37. 35.]
 [81. 37. 35. 95.]
 [49. 37. 18.  4.]
 [37. 35.  4. 96.]
 [35. 95. 96. 78.]
 [ 6. 13. 77.  6.]
 [13. 51.  6. 83.]
 [51. 75. 83. 50.]
 [77.  6. 60.  8.]
 [ 6. 83.  8. 67.]
 [83. 50. 67. 21.]
 [60.  8.  1. 99.]
 [ 8. 67. 99. 96.]
 [67. 21. 96. 39.]]

image [[[[ 81.  98.  88.  37.]
   [124.  83. 265.  84.]
   [124. 157. 338. 116.]
   [ 49. 132. 114.  78.]]]


 [[[  6.  88. 160.  50.]
   [ 14. 256. 148. 127.]
   [ 56. 257. 174. 150.]
   [ 60.  29.  97.  39.]]]]
```

convolution class

In [8]:
```
class Convolution:
    # W: フィルター, b: バイアス
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 中間データ（backward時に使用）
        self.x = None
        self.col = None
        self.col_W = None

        # フィルター・バイアスパラメータの勾配
        self.dW = None
        self.db = None

    def forward(self, x):
        # FN: filter_number, C: channel, FH: filter_height, FW: filter_width
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        # 出力値のheight, width
        out_h = 1 + int((H + 2 * self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2 * self.pad - FW) / self.stride)

        # xを行列に変換
        col = im2col(x, FH, FW, self.stride, self.pad)
        # フィルターをxに合わせた行列に変換
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        # 計算のために変えた形式を戻す
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out

    def backward(self, dout):
```

```
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0, 2, 3, 1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        # dcolを画像データに変換
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx
```

pooling class

In [9]:
```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.arg_max = None

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        # xを行列に変換
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        # プーリングのサイズに合わせてリサイズ
        col = col.reshape(-1, self.pool_h*self.pool_w)

        # 行ごとに最大値を求める
        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        # 整形
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.arg_max = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)
```

```
        return dx
```

sinple convolution network class

In [10]:
```
class SimpleConvNet:
    # conv - relu - pool - affine - relu - affine - softmax
    def __init__(self, input_dim=(1, 28, 28), conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2))

        # 重みの初期化
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)

        # レイヤの生成
        self.layers = OrderedDict()
        self.layers['Conv1'] = layers.Convolution(self.params['W1'], self.params['b1'], conv_param['stride'], conv_param['pad'])
        self.layers['Relu1'] = layers.Relu()
        self.layers['Pool1'] = layers.Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = layers.Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = layers.Relu()
        self.layers['Affine2'] = layers.Affine(self.params['W3'], self.params['b3'])

        self.last_layer = layers.SoftmaxWithLoss()

    def predict(self, x):
        for key in self.layers.keys():
            x = self.layers[key].forward(x)
        return x

    def loss(self, x, d):
        y = self.predict(x)
        return self.last_layer.forward(y, d)
```

```
    def accuracy(self, x, d, batch_size=100):
        if d.ndim != 1 : d = np.argmax(d, axis=1)

        acc = 0.0

        for i in range(int(x.shape[0] / batch_size)):
            tx = x[i*batch_size:(i+1)*batch_size]
            td = d[i*batch_size:(i+1)*batch_size]
            y = self.predict(tx)
            y = np.argmax(y, axis=1)
            acc += np.sum(y == td)

        return acc / x.shape[0]

    def gradient(self, x, d):
        # forward
        self.loss(x, d)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)
        layers = list(self.layers.values())

        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)

        # 設定
        grad = {}
        grad['W1'], grad['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
        grad['W2'], grad['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
        grad['W3'], grad['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

        return grad
```

In [20]:
```
from common import optimizer

# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

print("データ読み込み完了")

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]


network = SimpleConvNet(input_dim=(1,28,28), conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)

optimizer = optimizer.Adam()

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10



for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    optimizer.update(network.params, grad)

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i+1) % plot_interval == 0:
```

```
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                 : ' + str(i+1) + '. 正答率(テスト) = ' + str(accr_test))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

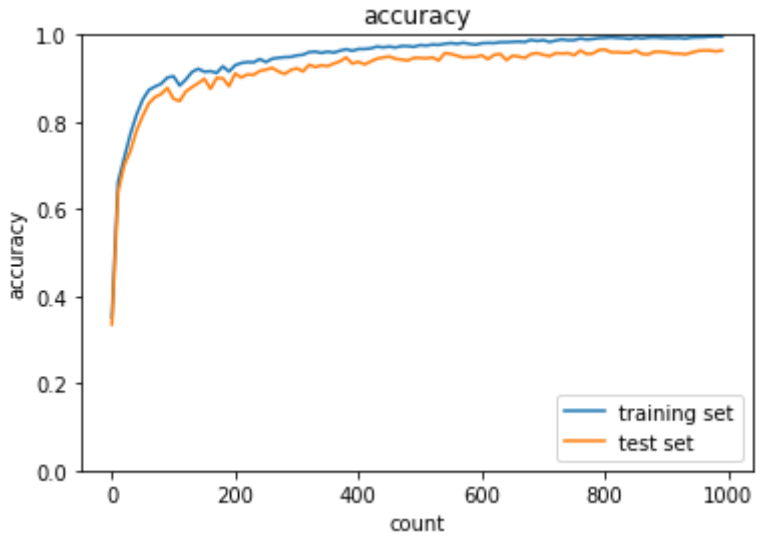
```
データ読み込み完了
Generation: 10. 正答率(トレーニング) = 0.3508
                : 10. 正答率(テスト) = 0.335
Generation: 20. 正答率(トレーニング) = 0.6624
                : 20. 正答率(テスト) = 0.639
Generation: 30. 正答率(トレーニング) = 0.7178
                : 30. 正答率(テスト) = 0.701
Generation: 40. 正答率(トレーニング) = 0.7724
                : 40. 正答率(テスト) = 0.733
Generation: 50. 正答率(トレーニング) = 0.8168
                : 50. 正答率(テスト) = 0.78
Generation: 60. 正答率(トレーニング) = 0.8514
                : 60. 正答率(テスト) = 0.814
Generation: 70. 正答率(トレーニング) = 0.8734
                : 70. 正答率(テスト) = 0.843
Generation: 80. 正答率(トレーニング) = 0.8812
                : 80. 正答率(テスト) = 0.857
Generation: 90. 正答率(トレーニング) = 0.8876
                : 90. 正答率(テスト) = 0.864
Generation: 100. 正答率(トレーニング) = 0.9018
                : 100. 正答率(テスト) = 0.878
Generation: 110. 正答率(トレーニング) = 0.9048
                : 110. 正答率(テスト) = 0.853
Generation: 120. 正答率(トレーニング) = 0.8834
                : 120. 正答率(テスト) = 0.848
Generation: 130. 正答率(トレーニング) = 0.8974
                : 130. 正答率(テスト) = 0.87
Generation: 140. 正答率(トレーニング) = 0.9148
                : 140. 正答率(テスト) = 0.88
Generation: 150. 正答率(トレーニング) = 0.922
                : 150. 正答率(テスト) = 0.889
Generation: 160. 正答率(トレーニング) = 0.9152
                : 160. 正答率(テスト) = 0.899
Generation: 170. 正答率(トレーニング) = 0.9164
                : 170. 正答率(テスト) = 0.876
Generation: 180. 正答率(トレーニング) = 0.9124
                : 180. 正答率(テスト) = 0.901
Generation: 190. 正答率(トレーニング) = 0.9276
                : 190. 正答率(テスト) = 0.9
Generation: 200. 正答率(トレーニング) = 0.9156
                : 200. 正答率(テスト) = 0.882
```

```
Generation: 210. 正答率(トレーニング) = 0.9302
                : 210. 正答率(テスト) = 0.911
Generation: 220. 正答率(トレーニング) = 0.9352
                : 220. 正答率(テスト) = 0.902
Generation: 230. 正答率(トレーニング) = 0.9374
                : 230. 正答率(テスト) = 0.909
Generation: 240. 正答率(トレーニング) = 0.9368
                : 240. 正答率(テスト) = 0.908
Generation: 250. 正答率(トレーニング) = 0.9442
                : 250. 正答率(テスト) = 0.917
Generation: 260. 正答率(トレーニング) = 0.9374
                : 260. 正答率(テスト) = 0.92
Generation: 270. 正答率(トレーニング) = 0.945
                : 270. 正答率(テスト) = 0.924
Generation: 280. 正答率(トレーニング) = 0.9472
                : 280. 正答率(テスト) = 0.916
Generation: 290. 正答率(トレーニング) = 0.9486
                : 290. 正答率(テスト) = 0.91
Generation: 300. 正答率(トレーニング) = 0.9492
                : 300. 正答率(テスト) = 0.919
Generation: 310. 正答率(トレーニング) = 0.9526
                : 310. 正答率(テスト) = 0.923
Generation: 320. 正答率(トレーニング) = 0.9542
                : 320. 正答率(テスト) = 0.916
Generation: 330. 正答率(トレーニング) = 0.9602
                : 330. 正答率(テスト) = 0.931
Generation: 340. 正答率(トレーニング) = 0.9614
                : 340. 正答率(テスト) = 0.926
Generation: 350. 正答率(トレーニング) = 0.9584
                : 350. 正答率(テスト) = 0.93
Generation: 360. 正答率(トレーニング) = 0.9614
                : 360. 正答率(テスト) = 0.928
Generation: 370. 正答率(トレーニング) = 0.9592
                : 370. 正答率(テスト) = 0.934
Generation: 380. 正答率(トレーニング) = 0.9634
                : 380. 正答率(テスト) = 0.939
Generation: 390. 正答率(トレーニング) = 0.967
                : 390. 正答率(テスト) = 0.948
Generation: 400. 正答率(トレーニング) = 0.9632
                : 400. 正答率(テスト) = 0.934
Generation: 410. 正答率(トレーニング) = 0.9674
                : 410. 正答率(テスト) = 0.938
```

```
Generation: 420. 正答率(トレーニング) = 0.9674
                 : 420. 正答率(テスト) = 0.932
Generation: 430. 正答率(トレーニング) = 0.9688
                 : 430. 正答率(テスト) = 0.939
Generation: 440. 正答率(トレーニング) = 0.973
                 : 440. 正答率(テスト) = 0.945
Generation: 450. 正答率(トレーニング) = 0.9712
                 : 450. 正答率(テスト) = 0.948
Generation: 460. 正答率(トレーニング) = 0.973
                 : 460. 正答率(テスト) = 0.95
Generation: 470. 正答率(トレーニング) = 0.9708
                 : 470. 正答率(テスト) = 0.945
Generation: 480. 正答率(トレーニング) = 0.9742
                 : 480. 正答率(テスト) = 0.943
Generation: 490. 正答率(トレーニング) = 0.974
                 : 490. 正答率(テスト) = 0.941
Generation: 500. 正答率(トレーニング) = 0.9724
                 : 500. 正答率(テスト) = 0.947
Generation: 510. 正答率(トレーニング) = 0.9764
                 : 510. 正答率(テスト) = 0.947
Generation: 520. 正答率(トレーニング) = 0.9752
                 : 520. 正答率(テスト) = 0.946
Generation: 530. 正答率(トレーニング) = 0.9776
                 : 530. 正答率(テスト) = 0.948
Generation: 540. 正答率(トレーニング) = 0.9768
                 : 540. 正答率(テスト) = 0.941
Generation: 550. 正答率(トレーニング) = 0.9792
                 : 550. 正答率(テスト) = 0.958
Generation: 560. 正答率(トレーニング) = 0.981
                 : 560. 正答率(テスト) = 0.956
Generation: 570. 正答率(トレーニング) = 0.9792
                 : 570. 正答率(テスト) = 0.952
Generation: 580. 正答率(トレーニング) = 0.9818
                 : 580. 正答率(テスト) = 0.948
Generation: 590. 正答率(トレーニング) = 0.9792
                 : 590. 正答率(テスト) = 0.949
Generation: 600. 正答率(トレーニング) = 0.9774
                 : 600. 正答率(テスト) = 0.949
Generation: 610. 正答率(トレーニング) = 0.9806
                 : 610. 正答率(テスト) = 0.953
Generation: 620. 正答率(トレーニング) = 0.9814
                 : 620. 正答率(テスト) = 0.944
```

```
Generation: 630. 正答率(トレーニング) = 0.981
                : 630. 正答率(テスト) = 0.954
Generation: 640. 正答率(トレーニング) = 0.983
                : 640. 正答率(テスト) = 0.956
Generation: 650. 正答率(トレーニング) = 0.983
                : 650. 正答率(テスト) = 0.942
Generation: 660. 正答率(トレーニング) = 0.9838
                : 660. 正答率(テスト) = 0.952
Generation: 670. 正答率(トレーニング) = 0.9844
                : 670. 正答率(テスト) = 0.95
Generation: 680. 正答率(トレーニング) = 0.9836
                : 680. 正答率(テスト) = 0.947
Generation: 690. 正答率(トレーニング) = 0.9882
                : 690. 正答率(テスト) = 0.956
Generation: 700. 正答率(トレーニング) = 0.9858
                : 700. 正答率(テスト) = 0.958
Generation: 710. 正答率(トレーニング) = 0.9876
                : 710. 正答率(テスト) = 0.954
Generation: 720. 正答率(トレーニング) = 0.9836
                : 720. 正答率(テスト) = 0.951
Generation: 730. 正答率(トレーニング) = 0.9868
                : 730. 正答率(テスト) = 0.958
Generation: 740. 正答率(トレーニング) = 0.9894
                : 740. 正答率(テスト) = 0.957
Generation: 750. 正答率(トレーニング) = 0.9876
                : 750. 正答率(テスト) = 0.958
Generation: 760. 正答率(トレーニング) = 0.9878
                : 760. 正答率(テスト) = 0.953
Generation: 770. 正答率(トレーニング) = 0.9916
                : 770. 正答率(テスト) = 0.964
Generation: 780. 正答率(トレーニング) = 0.9892
                : 780. 正答率(テスト) = 0.956
Generation: 790. 正答率(トレーニング) = 0.9904
                : 790. 正答率(テスト) = 0.957
Generation: 800. 正答率(トレーニング) = 0.9924
                : 800. 正答率(テスト) = 0.965
Generation: 810. 正答率(トレーニング) = 0.993
                : 810. 正答率(テスト) = 0.966
Generation: 820. 正答率(トレーニング) = 0.9942
                : 820. 正答率(テスト) = 0.96
Generation: 830. 正答率(トレーニング) = 0.993
                : 830. 正答率(テスト) = 0.96
```

```
Generation: 840. 正答率(トレーニング) = 0.9924
                 : 840. 正答率(テスト) = 0.959
Generation: 850. 正答率(トレーニング) = 0.9906
                 : 850. 正答率(テスト) = 0.959
Generation: 860. 正答率(トレーニング) = 0.9934
                 : 860. 正答率(テスト) = 0.964
Generation: 870. 正答率(トレーニング) = 0.9916
                 : 870. 正答率(テスト) = 0.956
Generation: 880. 正答率(トレーニング) = 0.9936
                 : 880. 正答率(テスト) = 0.955
Generation: 890. 正答率(トレーニング) = 0.9942
                 : 890. 正答率(テスト) = 0.961
Generation: 900. 正答率(トレーニング) = 0.9934
                 : 900. 正答率(テスト) = 0.961
Generation: 910. 正答率(トレーニング) = 0.9928
                 : 910. 正答率(テスト) = 0.96
Generation: 920. 正答率(トレーニング) = 0.9924
                 : 920. 正答率(テスト) = 0.957
Generation: 930. 正答率(トレーニング) = 0.9932
                 : 930. 正答率(テスト) = 0.957
Generation: 940. 正答率(トレーニング) = 0.9914
                 : 940. 正答率(テスト) = 0.955
Generation: 950. 正答率(トレーニング) = 0.9938
                 : 950. 正答率(テスト) = 0.959
Generation: 960. 正答率(トレーニング) = 0.9944
                 : 960. 正答率(テスト) = 0.963
Generation: 970. 正答率(トレーニング) = 0.9946
                 : 970. 正答率(テスト) = 0.964
Generation: 980. 正答率(トレーニング) = 0.996
                 : 980. 正答率(テスト) = 0.964
Generation: 990. 正答率(トレーニング) = 0.9964
                 : 990. 正答率(テスト) = 0.962
Generation: 1000. 正答率(トレーニング) = 0.9958
                 : 1000. 正答率(テスト) = 0.964
```


accuracy
1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
200
400
600
800
1000
count
training set
test set

試行

重みの変更

In [17]:
```
from common import optimizer

# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

print("データ読み込み完了")

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]


network = SimpleConvNet(input_dim=(1,28,28), conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=1)

optimizer = optimizer.Adam()

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10



for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    optimizer.update(network.params, grad)

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i+1) % plot_interval == 0:
```

```
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + str(accr_test))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

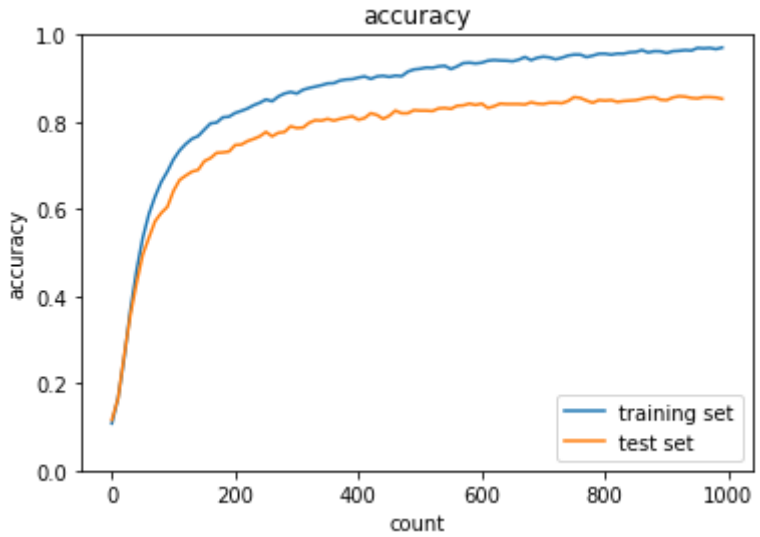
```
データ読み込み完了
Generation: 10. 正答率(トレーニング) = 0.108
                : 10. 正答率(テスト) = 0.116
Generation: 20. 正答率(トレーニング) = 0.1662
                : 20. 正答率(テスト) = 0.165
Generation: 30. 正答率(トレーニング) = 0.2584
                : 30. 正答率(テスト) = 0.264
Generation: 40. 正答率(トレーニング) = 0.3678
                : 40. 正答率(テスト) = 0.357
Generation: 50. 正答率(トレーニング) = 0.459
                : 50. 正答率(テスト) = 0.43
Generation: 60. 正答率(トレーニング) = 0.5336
                : 60. 正答率(テスト) = 0.495
Generation: 70. 正答率(トレーニング) = 0.5884
                : 70. 正答率(テスト) = 0.534
Generation: 80. 正答率(トレーニング) = 0.6286
                : 80. 正答率(テスト) = 0.572
Generation: 90. 正答率(トレーニング) = 0.662
                : 90. 正答率(テスト) = 0.591
Generation: 100. 正答率(トレーニング) = 0.6862
                : 100. 正答率(テスト) = 0.605
Generation: 110. 正答率(トレーニング) = 0.7142
                : 110. 正答率(テスト) = 0.642
Generation: 120. 正答率(トレーニング) = 0.7348
                : 120. 正答率(テスト) = 0.667
Generation: 130. 正答率(トレーニング) = 0.7496
                : 130. 正答率(テスト) = 0.677
Generation: 140. 正答率(トレーニング) = 0.7614
                : 140. 正答率(テスト) = 0.686
Generation: 150. 正答率(トレーニング) = 0.7676
                : 150. 正答率(テスト) = 0.69
Generation: 160. 正答率(トレーニング) = 0.7824
                : 160. 正答率(テスト) = 0.71
Generation: 170. 正答率(トレーニング) = 0.7966
                : 170. 正答率(テスト) = 0.717
Generation: 180. 正答率(トレーニング) = 0.7986
                : 180. 正答率(テスト) = 0.729
Generation: 190. 正答率(トレーニング) = 0.8106
                : 190. 正答率(テスト) = 0.73
Generation: 200. 正答率(トレーニング) = 0.812
                : 200. 正答率(テスト) = 0.732
```

```
Generation: 210. 正答率(トレーニング) = 0.8206
                : 210. 正答率(テスト) = 0.747
Generation: 220. 正答率(トレーニング) = 0.8254
                : 220. 正答率(テスト) = 0.748
Generation: 230. 正答率(トレーニング) = 0.8308
                : 230. 正答率(テスト) = 0.756
Generation: 240. 正答率(トレーニング) = 0.8384
                : 240. 正答率(テスト) = 0.761
Generation: 250. 正答率(トレーニング) = 0.844
                : 250. 正答率(テスト) = 0.767
Generation: 260. 正答率(トレーニング) = 0.8516
                : 260. 正答率(テスト) = 0.777
Generation: 270. 正答率(トレーニング) = 0.8474
                : 270. 正答率(テスト) = 0.767
Generation: 280. 正答率(トレーニング) = 0.8584
                : 280. 正答率(テスト) = 0.775
Generation: 290. 正答率(トレーニング) = 0.8652
                : 290. 正答率(テスト) = 0.777
Generation: 300. 正答率(トレーニング) = 0.8688
                : 300. 正答率(テスト) = 0.79
Generation: 310. 正答率(トレーニング) = 0.8652
                : 310. 正答率(テスト) = 0.786
Generation: 320. 正答率(トレーニング) = 0.8738
                : 320. 正答率(テスト) = 0.787
Generation: 330. 正答率(トレーニング) = 0.8778
                : 330. 正答率(テスト) = 0.798
Generation: 340. 正答率(トレーニング) = 0.881
                : 340. 正答率(テスト) = 0.804
Generation: 350. 正答率(トレーニング) = 0.8842
                : 350. 正答率(テスト) = 0.803
Generation: 360. 正答率(トレーニング) = 0.8884
                : 360. 正答率(テスト) = 0.807
Generation: 370. 正答率(トレーニング) = 0.889
                : 370. 正答率(テスト) = 0.803
Generation: 380. 正答率(トレーニング) = 0.895
                : 380. 正答率(テスト) = 0.807
Generation: 390. 正答率(トレーニング) = 0.8974
                : 390. 正答率(テスト) = 0.81
Generation: 400. 正答率(トレーニング) = 0.8982
                : 400. 正答率(テスト) = 0.813
Generation: 410. 正答率(トレーニング) = 0.9018
                : 410. 正答率(テスト) = 0.805
```

```
Generation: 420. 正答率(トレーニング) = 0.9044
                 : 420. 正答率(テスト) = 0.809
Generation: 430. 正答率(トレーニング) = 0.8988
                 : 430. 正答率(テスト) = 0.82
Generation: 440. 正答率(トレーニング) = 0.9044
                 : 440. 正答率(テスト) = 0.816
Generation: 450. 正答率(トレーニング) = 0.9058
                 : 450. 正答率(テスト) = 0.807
Generation: 460. 正答率(トレーニング) = 0.9034
                 : 460. 正答率(テスト) = 0.814
Generation: 470. 正答率(トレーニング) = 0.9062
                 : 470. 正答率(テスト) = 0.826
Generation: 480. 正答率(トレーニング) = 0.905
                 : 480. 正答率(テスト) = 0.82
Generation: 490. 正答率(トレーニング) = 0.9154
                 : 490. 正答率(テスト) = 0.82
Generation: 500. 正答率(トレーニング) = 0.9204
                 : 500. 正答率(テスト) = 0.827
Generation: 510. 正答率(トレーニング) = 0.922
                 : 510. 正答率(テスト) = 0.826
Generation: 520. 正答率(トレーニング) = 0.9248
                 : 520. 正答率(テスト) = 0.826
Generation: 530. 正答率(トレーニング) = 0.9242
                 : 530. 正答率(テスト) = 0.825
Generation: 540. 正答率(トレーニング) = 0.9274
                 : 540. 正答率(テスト) = 0.831
Generation: 550. 正答率(トレーニング) = 0.9288
                 : 550. 正答率(テスト) = 0.832
Generation: 560. 正答率(トレーニング) = 0.9214
                 : 560. 正答率(テスト) = 0.83
Generation: 570. 正答率(トレーニング) = 0.9274
                 : 570. 正答率(テスト) = 0.837
Generation: 580. 正答率(トレーニング) = 0.935
                 : 580. 正答率(テスト) = 0.838
Generation: 590. 正答率(トレーニング) = 0.936
                 : 590. 正答率(テスト) = 0.842
Generation: 600. 正答率(トレーニング) = 0.9342
                 : 600. 正答率(テスト) = 0.839
Generation: 610. 正答率(トレーニング) = 0.936
                 : 610. 正答率(テスト) = 0.842
Generation: 620. 正答率(トレーニング) = 0.9406
                 : 620. 正答率(テスト) = 0.832
```

```
Generation: 630. 正答率(トレーニング) = 0.942
                : 630. 正答率(テスト) = 0.836
Generation: 640. 正答率(トレーニング) = 0.941
                : 640. 正答率(テスト) = 0.842
Generation: 650. 正答率(トレーニング) = 0.9406
                : 650. 正答率(テスト) = 0.841
Generation: 660. 正答率(トレーニング) = 0.9392
                : 660. 正答率(テスト) = 0.841
Generation: 670. 正答率(トレーニング) = 0.9428
                : 670. 正答率(テスト) = 0.841
Generation: 680. 正答率(トレーニング) = 0.949
                : 680. 正答率(テスト) = 0.84
Generation: 690. 正答率(トレーニング) = 0.9418
                : 690. 正答率(テスト) = 0.845
Generation: 700. 正答率(トレーニング) = 0.9472
                : 700. 正答率(テスト) = 0.842
Generation: 710. 正答率(トレーニング) = 0.9498
                : 710. 正答率(テスト) = 0.841
Generation: 720. 正答率(トレーニング) = 0.948
                : 720. 正答率(テスト) = 0.844
Generation: 730. 正答率(トレーニング) = 0.9438
                : 730. 正答率(テスト) = 0.844
Generation: 740. 正答率(トレーニング) = 0.9474
                : 740. 正答率(テスト) = 0.843
Generation: 750. 正答率(トレーニング) = 0.9522
                : 750. 正答率(テスト) = 0.848
Generation: 760. 正答率(トレーニング) = 0.9546
                : 760. 正答率(テスト) = 0.857
Generation: 770. 正答率(トレーニング) = 0.9544
                : 770. 正答率(テスト) = 0.855
Generation: 780. 正答率(トレーニング) = 0.9488
                : 780. 正答率(テスト) = 0.849
Generation: 790. 正答率(トレーニング) = 0.9522
                : 790. 正答率(テスト) = 0.844
Generation: 800. 正答率(トレーニング) = 0.9564
                : 800. 正答率(テスト) = 0.85
Generation: 810. 正答率(トレーニング) = 0.9566
                : 810. 正答率(テスト) = 0.849
Generation: 820. 正答率(トレーニング) = 0.9544
                : 820. 正答率(テスト) = 0.85
Generation: 830. 正答率(トレーニング) = 0.9568
                : 830. 正答率(テスト) = 0.846
```

```
Generation: 840. 正答率(トレーニング) = 0.9566
                 : 840. 正答率(テスト) = 0.848
Generation: 850. 正答率(トレーニング) = 0.9598
                 : 850. 正答率(テスト) = 0.849
Generation: 860. 正答率(トレーニング) = 0.9606
                 : 860. 正答率(テスト) = 0.85
Generation: 870. 正答率(トレーニング) = 0.9648
                 : 870. 正答率(テスト) = 0.853
Generation: 880. 正答率(トレーニング) = 0.9594
                 : 880. 正答率(テスト) = 0.856
Generation: 890. 正答率(トレーニング) = 0.9624
                 : 890. 正答率(テスト) = 0.857
Generation: 900. 正答率(トレーニング) = 0.962
                 : 900. 正答率(テスト) = 0.851
Generation: 910. 正答率(トレーニング) = 0.9582
                 : 910. 正答率(テスト) = 0.85
Generation: 920. 正答率(トレーニング) = 0.9624
                 : 920. 正答率(テスト) = 0.856
Generation: 930. 正答率(トレーニング) = 0.9632
                 : 930. 正答率(テスト) = 0.859
Generation: 940. 正答率(トレーニング) = 0.9648
                 : 940. 正答率(テスト) = 0.858
Generation: 950. 正答率(トレーニング) = 0.9638
                 : 950. 正答率(テスト) = 0.855
Generation: 960. 正答率(トレーニング) = 0.9698
                 : 960. 正答率(テスト) = 0.854
Generation: 970. 正答率(トレーニング) = 0.9688
                 : 970. 正答率(テスト) = 0.857
Generation: 980. 正答率(トレーニング) = 0.9698
                 : 980. 正答率(テスト) = 0.857
Generation: 990. 正答率(トレーニング) = 0.9672
                 : 990. 正答率(テスト) = 0.856
Generation: 1000. 正答率(トレーニング) = 0.9706
                 : 1000. 正答率(テスト) = 0.853
```



試行

Optimizerの変更（Adam ⇒ SGD）

In [14]:
```
from common import optimizer

# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

print("データ読み込み完了")

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]


network = SimpleConvNet(input_dim=(1,28,28), conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)

optimizer = optimizer.SGD(0.1)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10



for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    optimizer.update(network.params, grad)

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i+1) % plot_interval == 0:
```

```
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                 : ' + str(i+1) + '. 正答率(テスト) = ' + str(accr_test))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

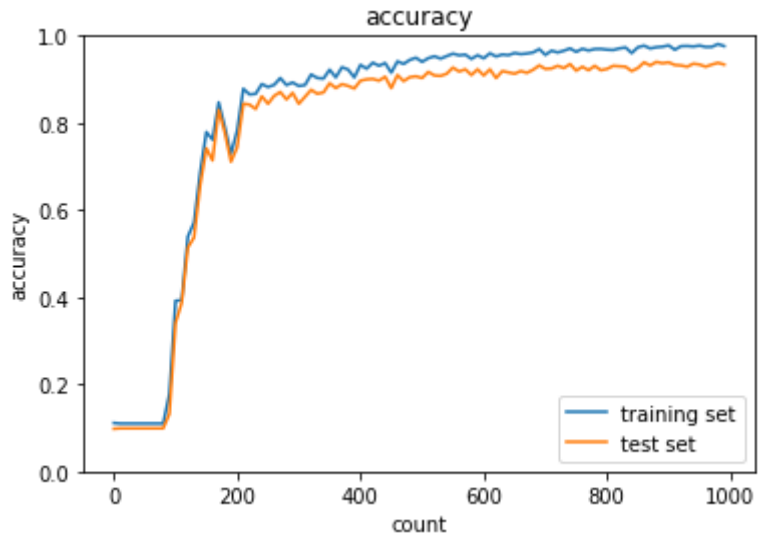
```
データ読み込み完了
Generation: 10. 正答率(トレーニング) = 0.1112
                : 10. 正答率(テスト) = 0.098
Generation: 20. 正答率(トレーニング) = 0.11
                : 20. 正答率(テスト) = 0.099
Generation: 30. 正答率(トレーニング) = 0.11
                : 30. 正答率(テスト) = 0.099
Generation: 40. 正答率(トレーニング) = 0.11
                : 40. 正答率(テスト) = 0.099
Generation: 50. 正答率(トレーニング) = 0.11
                : 50. 正答率(テスト) = 0.099
Generation: 60. 正答率(トレーニング) = 0.11
                : 60. 正答率(テスト) = 0.099
Generation: 70. 正答率(トレーニング) = 0.11
                : 70. 正答率(テスト) = 0.099
Generation: 80. 正答率(トレーニング) = 0.11
                : 80. 正答率(テスト) = 0.099
Generation: 90. 正答率(トレーニング) = 0.11
                : 90. 正答率(テスト) = 0.099
Generation: 100. 正答率(トレーニング) = 0.178
                : 100. 正答率(テスト) = 0.132
Generation: 110. 正答率(トレーニング) = 0.3918
                : 110. 正答率(テスト) = 0.341
Generation: 120. 正答率(トレーニング) = 0.3928
                : 120. 正答率(テスト) = 0.387
Generation: 130. 正答率(トレーニング) = 0.5392
                : 130. 正答率(テスト) = 0.514
Generation: 140. 正答率(トレーニング) = 0.5714
                : 140. 正答率(テスト) = 0.537
Generation: 150. 正答率(トレーニング) = 0.6866
                : 150. 正答率(テスト) = 0.657
Generation: 160. 正答率(トレーニング) = 0.779
                : 160. 正答率(テスト) = 0.742
Generation: 170. 正答率(トレーニング) = 0.7618
                : 170. 正答率(テスト) = 0.714
Generation: 180. 正答率(トレーニング) = 0.8472
                : 180. 正答率(テスト) = 0.83
Generation: 190. 正答率(トレーニング) = 0.7906
                : 190. 正答率(テスト) = 0.778
Generation: 200. 正答率(トレーニング) = 0.729
                : 200. 正答率(テスト) = 0.711
```

```
Generation: 210. 正答率(トレーニング) = 0.7824
                : 210. 正答率(テスト) = 0.744
Generation: 220. 正答率(トレーニング) = 0.8784
                : 220. 正答率(テスト) = 0.844
Generation: 230. 正答率(トレーニング) = 0.8654
                : 230. 正答率(テスト) = 0.842
Generation: 240. 正答率(トレーニング) = 0.8668
                : 240. 正答率(テスト) = 0.832
Generation: 250. 正答率(トレーニング) = 0.8892
                : 250. 正答率(テスト) = 0.861
Generation: 260. 正答率(トレーニング) = 0.8824
                : 260. 正答率(テスト) = 0.844
Generation: 270. 正答率(トレーニング) = 0.8874
                : 270. 正答率(テスト) = 0.862
Generation: 280. 正答率(トレーニング) = 0.903
                : 280. 正答率(テスト) = 0.871
Generation: 290. 正答率(トレーニング) = 0.887
                : 290. 正答率(テスト) = 0.854
Generation: 300. 正答率(トレーニング) = 0.8928
                : 300. 正答率(テスト) = 0.869
Generation: 310. 正答率(トレーニング) = 0.8848
                : 310. 正答率(テスト) = 0.844
Generation: 320. 正答率(トレーニング) = 0.887
                : 320. 正答率(テスト) = 0.86
Generation: 330. 正答率(トレーニング) = 0.9112
                : 330. 正答率(テスト) = 0.876
Generation: 340. 正答率(トレーニング) = 0.9032
                : 340. 正答率(テスト) = 0.867
Generation: 350. 正答率(トレーニング) = 0.9018
                : 350. 正答率(テスト) = 0.87
Generation: 360. 正答率(トレーニング) = 0.9222
                : 360. 正答率(テスト) = 0.891
Generation: 370. 正答率(トレーニング) = 0.9046
                : 370. 正答率(テスト) = 0.88
Generation: 380. 正答率(トレーニング) = 0.9278
                : 380. 正答率(テスト) = 0.889
Generation: 390. 正答率(トレーニング) = 0.9226
                : 390. 正答率(テスト) = 0.885
Generation: 400. 正答率(トレーニング) = 0.9044
                : 400. 正答率(テスト) = 0.879
Generation: 410. 正答率(トレーニング) = 0.9332
                : 410. 正答率(テスト) = 0.897
```

```
Generation: 420. 正答率(トレーニング) = 0.9246
                 : 420. 正答率(テスト) = 0.9
Generation: 430. 正答率(トレーニング) = 0.9382
                 : 430. 正答率(テスト) = 0.901
Generation: 440. 正答率(トレーニング) = 0.9312
                 : 440. 正答率(テスト) = 0.897
Generation: 450. 正答率(トレーニング) = 0.9374
                 : 450. 正答率(テスト) = 0.906
Generation: 460. 正答率(トレーニング) = 0.9154
                 : 460. 正答率(テスト) = 0.88
Generation: 470. 正答率(トレーニング) = 0.9414
                 : 470. 正答率(テスト) = 0.91
Generation: 480. 正答率(トレーニング) = 0.9354
                 : 480. 正答率(テスト) = 0.896
Generation: 490. 正答率(トレーニング) = 0.9446
                 : 490. 正答率(テスト) = 0.905
Generation: 500. 正答率(トレーニング) = 0.9496
                 : 500. 正答率(テスト) = 0.907
Generation: 510. 正答率(トレーニング) = 0.9402
                 : 510. 正答率(テスト) = 0.903
Generation: 520. 正答率(トレーニング) = 0.9486
                 : 520. 正答率(テスト) = 0.917
Generation: 530. 正答率(トレーニング) = 0.953
                 : 530. 正答率(テスト) = 0.909
Generation: 540. 正答率(トレーニング) = 0.9476
                 : 540. 正答率(テスト) = 0.908
Generation: 550. 正答率(トレーニング) = 0.9534
                 : 550. 正答率(テスト) = 0.913
Generation: 560. 正答率(トレーニング) = 0.959
                 : 560. 正答率(テスト) = 0.927
Generation: 570. 正答率(トレーニング) = 0.9552
                 : 570. 正答率(テスト) = 0.918
Generation: 580. 正答率(トレーニング) = 0.9568
                 : 580. 正答率(テスト) = 0.923
Generation: 590. 正答率(トレーニング) = 0.947
                 : 590. 正答率(テスト) = 0.91
Generation: 600. 正答率(トレーニング) = 0.9554
                 : 600. 正答率(テスト) = 0.923
Generation: 610. 正答率(トレーニング) = 0.9484
                 : 610. 正答率(テスト) = 0.909
Generation: 620. 正答率(トレーニング) = 0.9598
                 : 620. 正答率(テスト) = 0.923
```

```
Generation: 630. 正答率(トレーニング) = 0.9526
                : 630. 正答率(テスト) = 0.903
Generation: 640. 正答率(トレーニング) = 0.957
                : 640. 正答率(テスト) = 0.919
Generation: 650. 正答率(トレーニング) = 0.9556
                : 650. 正答率(テスト) = 0.916
Generation: 660. 正答率(トレーニング) = 0.9604
                : 660. 正答率(テスト) = 0.913
Generation: 670. 正答率(トレーニング) = 0.958
                : 670. 正答率(テスト) = 0.919
Generation: 680. 正答率(トレーニング) = 0.9594
                : 680. 正答率(テスト) = 0.915
Generation: 690. 正答率(トレーニング) = 0.9618
                : 690. 正答率(テスト) = 0.922
Generation: 700. 正答率(トレーニング) = 0.9694
                : 700. 正答率(テスト) = 0.932
Generation: 710. 正答率(トレーニング) = 0.9562
                : 710. 正答率(テスト) = 0.924
Generation: 720. 正答率(トレーニング) = 0.9658
                : 720. 正答率(テスト) = 0.925
Generation: 730. 正答率(トレーニング) = 0.9618
                : 730. 正答率(テスト) = 0.931
Generation: 740. 正答率(トレーニング) = 0.9658
                : 740. 正答率(テスト) = 0.926
Generation: 750. 正答率(トレーニング) = 0.971
                : 750. 正答率(テスト) = 0.935
Generation: 760. 正答率(トレーニング) = 0.9632
                : 760. 正答率(テスト) = 0.92
Generation: 770. 正答率(トレーニング) = 0.9702
                : 770. 正答率(テスト) = 0.929
Generation: 780. 正答率(トレーニング) = 0.9662
                : 780. 正答率(テスト) = 0.921
Generation: 790. 正答率(トレーニング) = 0.9696
                : 790. 正答率(テスト) = 0.931
Generation: 800. 正答率(トレーニング) = 0.9698
                : 800. 正答率(テスト) = 0.921
Generation: 810. 正答率(トレーニング) = 0.9682
                : 810. 正答率(テスト) = 0.924
Generation: 820. 正答率(トレーニング) = 0.9674
                : 820. 正答率(テスト) = 0.931
Generation: 830. 正答率(トレーニング) = 0.971
                : 830. 正答率(テスト) = 0.93
```

```
Generation: 840. 正答率(トレーニング) = 0.9732
                 : 840. 正答率(テスト) = 0.929
Generation: 850. 正答率(トレーニング) = 0.9604
                 : 850. 正答率(テスト) = 0.919
Generation: 860. 正答率(トレーニング) = 0.9736
                 : 860. 正答率(テスト) = 0.926
Generation: 870. 正答率(トレーニング) = 0.9774
                 : 870. 正答率(テスト) = 0.939
Generation: 880. 正答率(トレーニング) = 0.971
                 : 880. 正答率(テスト) = 0.932
Generation: 890. 正答率(トレーニング) = 0.9738
                 : 890. 正答率(テスト) = 0.94
Generation: 900. 正答率(トレーニング) = 0.975
                 : 900. 正答率(テスト) = 0.937
Generation: 910. 正答率(トレーニング) = 0.978
                 : 910. 正答率(テスト) = 0.939
Generation: 920. 正答率(トレーニング) = 0.968
                 : 920. 正答率(テスト) = 0.933
Generation: 930. 正答率(トレーニング) = 0.9762
                 : 930. 正答率(テスト) = 0.932
Generation: 940. 正答率(トレーニング) = 0.9766
                 : 940. 正答率(テスト) = 0.929
Generation: 950. 正答率(トレーニング) = 0.9754
                 : 950. 正答率(テスト) = 0.936
Generation: 960. 正答率(トレーニング) = 0.9776
                 : 960. 正答率(テスト) = 0.934
Generation: 970. 正答率(トレーニング) = 0.9744
                 : 970. 正答率(テスト) = 0.929
Generation: 980. 正答率(トレーニング) = 0.9746
                 : 980. 正答率(テスト) = 0.934
Generation: 990. 正答率(トレーニング) = 0.981
                 : 990. 正答率(テスト) = 0.938
Generation: 1000. 正答率(トレーニング) = 0.9764
                 : 1000. 正答率(テスト) = 0.934
```


accuracy
1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
200
400
600
800
1000
count
training set
test set