Pythonによるプログラム説明

知的高性能計算研究室

呂氷（ロビン）

画像の読み込みと書き込み

- PILにおけるImage関数を利用

```
from PIL import Image

img = Image.open('example.jpg')
img.show()
img.save('save.jpg')
```

読み込み

書き込み

- Opencvを利用

```
import cv2

img = cv2.imread('example.jpg')
cv2.imshow('show this image',img)
cv2.imwrite('save.jpg',img)
```

読み込み

書き込み

example.jpg

example.jpg

save.jpg

画像サイズの変化

- 画像サイズを(1500,1000)の固定値とする(PIL)

```python
from PIL import Image

img = Image.open('example.jpg')
print ('the size of this image is: ', img.size)
img_resize = img.resize((1500, 1000))
print ('the changed image size is: ', img_resize.size)
img_resize.save('resize.jpg')
```

```
ihpc@ihpc-OptiPlex-9020:~/python-explain$ python size.py
('the size of this image is: ', (3060, 2146))
('the changed image size is: ', (1500, 1000))
```

- 画像サイズを0.3倍に縮小(Opencv)

```python
import cv2

img1 = cv2.imread('example.jpg',1)
print ('the size of original image is: ', img1.shape)
img1_resize = cv2.resize(img1, None, fx=0.3, fy=0.3, interpolation=cv2.INTER_AREA)
print ('the changed original image size is: ', img1_resize.shape)
```

RGB

```
ihpc@ihpc-OptiPlex-9020:~/python-explain$ python reduce.py
('the size of image is: ', (2146, 3060, 3))
('the reduced 0.3 times image size is: ', (644, 918, 3))
```

画像の回転

- 回転画像90度(PIL)

```
from PIL import Image

img = Image.open('example.jpg')
img_rotate = img.rotate(90,expand = True)
img_rotate.save('rotate.jpg')
```

- 回転画像90度(Opencv)

```
import cv2

img=cv2.imread('example.jpg',1)
img90=np.rot90(img)
cv2.imwrite("rotate.jpg",img90)
```

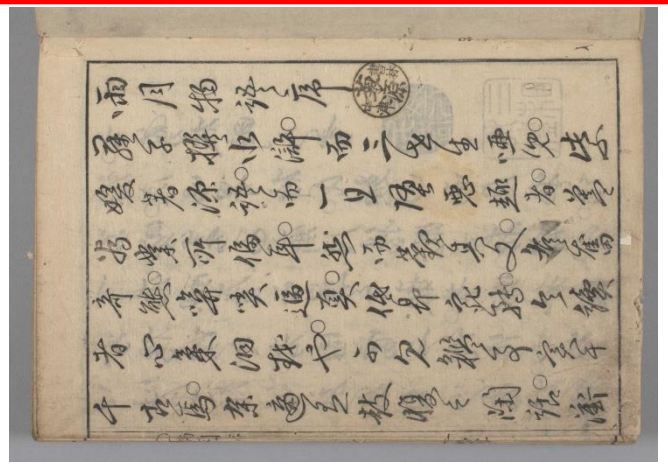


図１　原図

方向

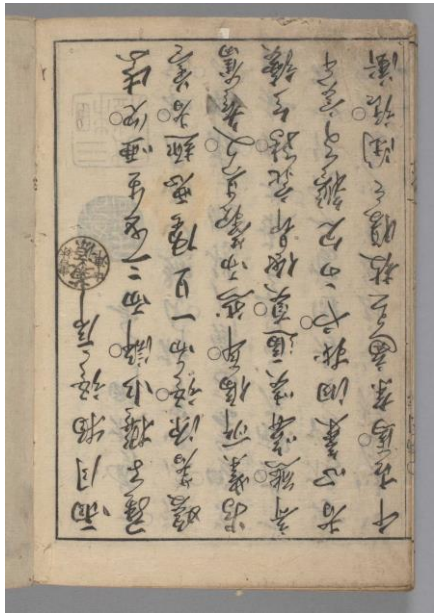


図２　回転画像

グレースケール画像に変換

- PIL

```
from PIL import Image

img = Image.open('example.jpg')
img.show()
img_gray = img.convert('L')
img_gray.save('gray-PIL.jpg')
```

- Opencv

```
import cv2

img = cv2.imread('example.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite('gray-opencv.jpg',img_gray)
```

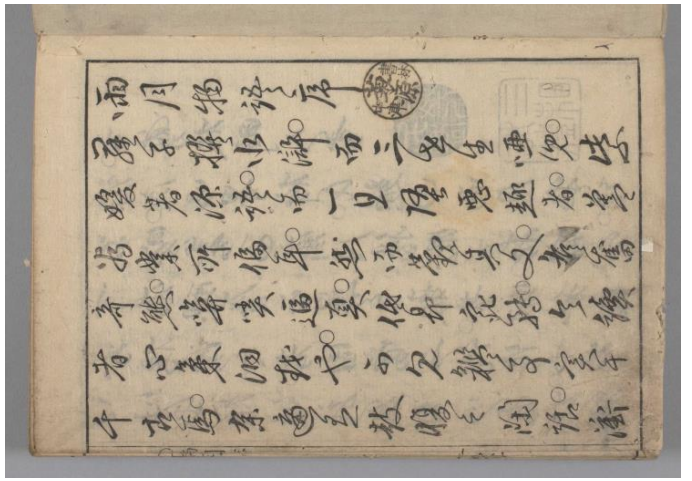


図１　原図

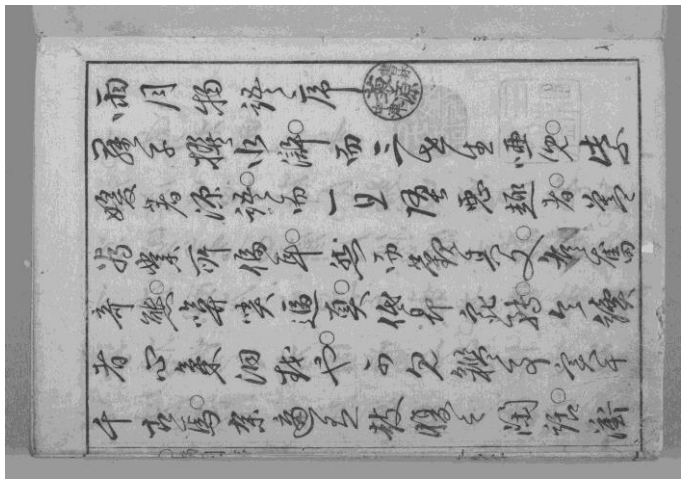


図２　グレースケール画像

二値化画像に変換

➢ 二値化画像(PIL)

```
from PIL import Image

img = Image.open('example.jpg')
img_gray = img.convert('L')
img_gray.save("img_gray.jpg")

def filter(i):
    thr = 127
    if i > thr:
        return 255
    else :
        return 0
img_thr = img_gray.point(filter)
img_thr.save('img-threshold.jpg')
```

グレースケール化

➢ 二値化画像(Opencv)

```
import cv2

img = cv2.imread("example.jpg")
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thr = cv2.threshold(img_gray, 127, 225, cv2.THRESH_BINARY)
cv2.imwrite('img-threshold.jpg', thr)
```

グレースケール化

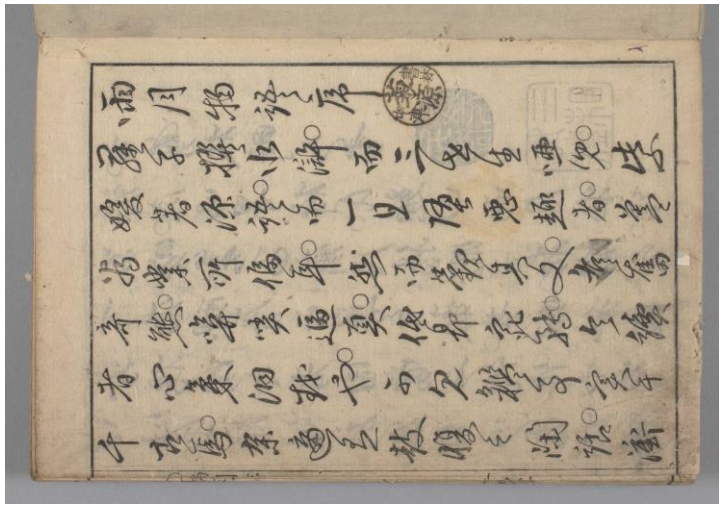


図１　原図

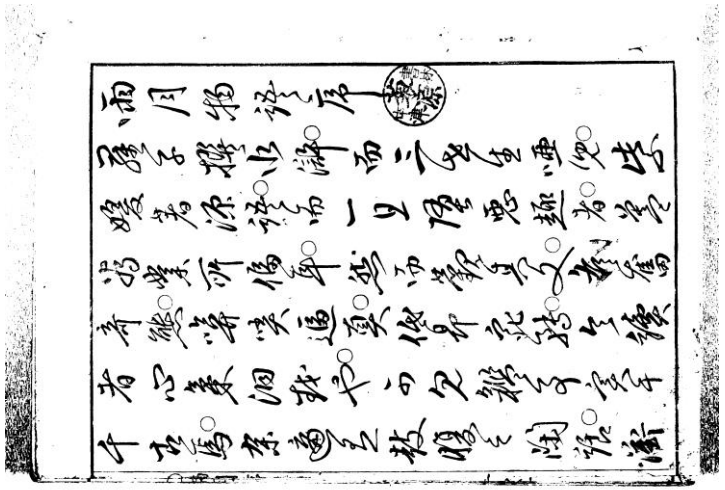


図２　二値化画像

画像のピクセル計算

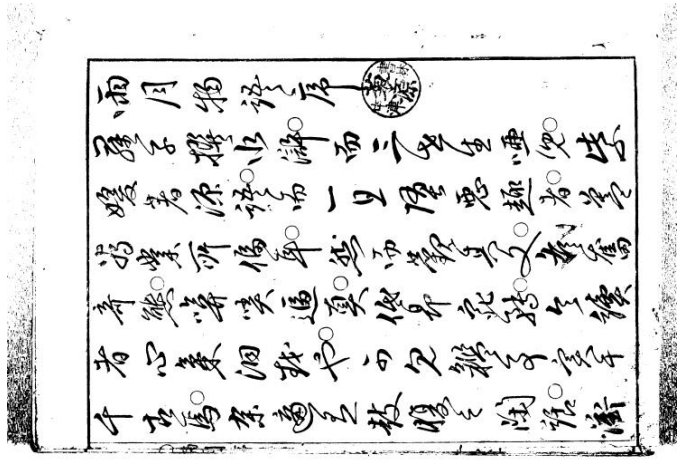
```
img = Image.open('example.jpg')
img_gray = img.convert('L')
img_gray.save("img_gray.jpg")

def filter(i):
    thr = 127
    if i > thr:
        return 255
    else :
        return 0
img_thr = img_gray.point(filter)
img_thr.save('img_thr.jpg')

img_rgb = img_thr.convert('RGB')
size = img_rgb.size
print 'the number of pixels on X axis:',size [0]
print 'the number of pixels on Y axis:',size [1]
hor = [0] *  size [0]
ver = [0] *  size [1]
pixel = size[0] * size[1]
print 'the pixels of this image:',pixel

for x in range (size [0]):
    for y in range(size [1]):
        r,g,b = img_rgb.getpixel((x, y))
        if r == 255 or g == 255 or b == 255:
            hor[x] =  hor[x] + 1
            ver[y] =  ver[y] + 1
y = np.array(hor)
plt.figure(figsize=(10, 4))
plt.plot(np.arange(len(y)), y)
plt.show()
```

二値化

hor: X軸
ver: Y軸

```
ihpc@ihpc-OptiPlex-9020:~/python-explain$ python pixel.py
the number of pixels on X axis: 3060
the number of pixels on Y axis: 2146
the pixels of this image: 6566760
```

図１　プログラム実行結果



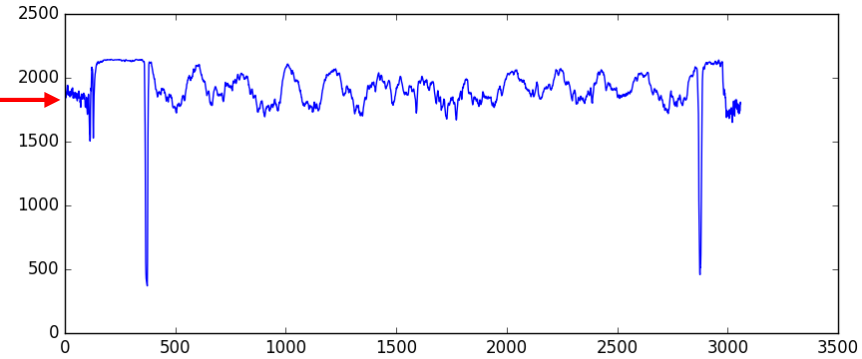


図２ピクセル計算（X軸）

バッチ処理(PIL)

```
import os
import os, sys
from PIL import Image
import numpy as np

path1 = './batch/'
path2 = './batch_gray/'

files = os.listdir(path1)
n = len(files)
print 'the files: ',files

for Fir in range (0,n):
    path3 = path1 + files[Fir] + '/'
    imgs = os.listdir(path3)
    print 'the images: ', imgs
    m = len(imgs)
    path4 = path2 + files[Fir] + '/'
    os.path.join(path2, files[Fir])
    if(not(os.path.exists(path4))):
        os.mkdir(path4)

    for Sec in range (0,m):
        path5 = path3 + imgs[Sec]
        path6 = path4 + imgs[Sec]
        image = Image.open(path5)
        image_gray = image.convert('L')
        image_gray.save(path6)
```

指定されたフォルダに含まれるフォルダ名のリストを返す

フォルダが存在しない場合はフォルダを作成

画像処理

図１　フォルダ構造

```
ihpc@ihpc-OptiPlex-9020:~/python-explain$ python b
the files:  ['file1', 'file2', 'file3']
the images:  ['img1.jpg']
the images:  ['img3.jpg', 'img2.jpg']
the images:  ['img6.jpg', 'img5.jpg', 'img4.jpg']
```

図2　プログラム実行結果

研究室中間発表

知的高性能計算研究室

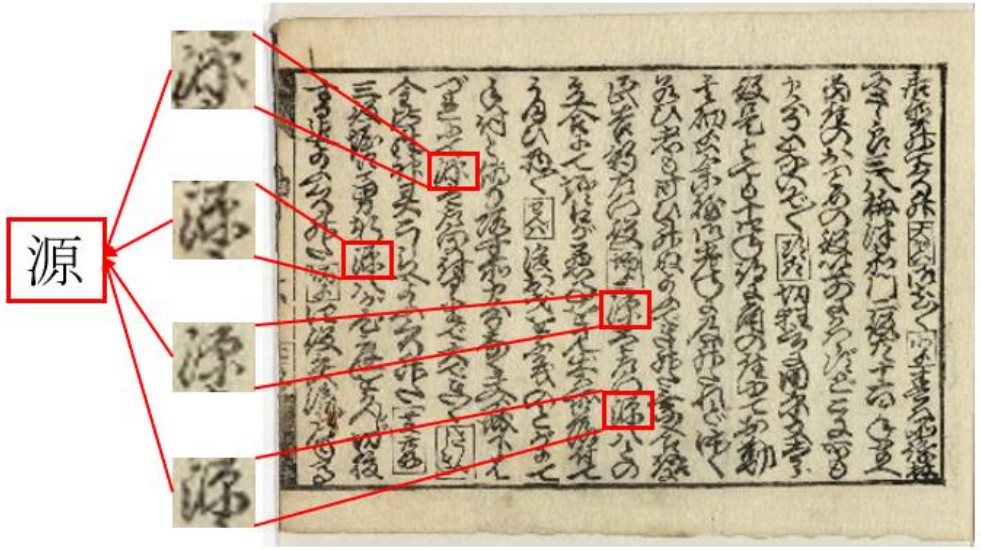
呂氷（ロビン）

目次

- 研究背景･目的
- 提案システム
  - 前処理
  - テキスト検出
  - テキスト切り取り
  - 文字認識
- 実験条件
- 実験結果
- まとめと今後の課題

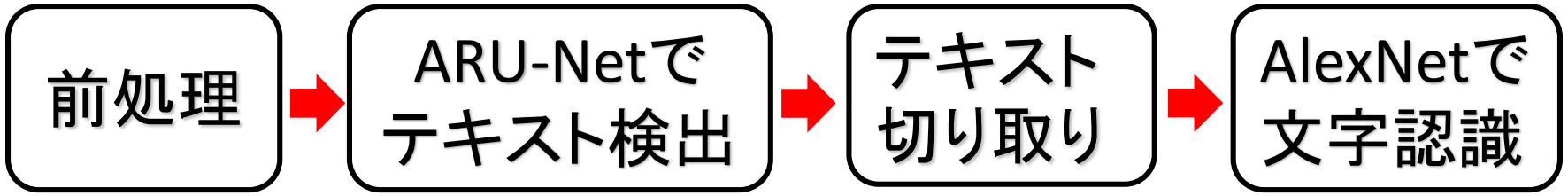
研究背景

- 研究者らは深層学習を用いて、くずし文章の自動認識を目指しているが、難航している
  - くずし文字のバリエーションが多い
  - 文字同士のつながり
  - 文章と絵が混在



研究目的

- 画像処理と深層学習を用いて、日本古典籍の画像に対して、文章領域の抽出から文字認識までの自動化を目指し、日本古典籍の解読整理と保護に貢献



提案した文字認識システムの流れ

前処理（1/2）：枠除去

原図

↓

画像二値化

↓

ピクセル計算

↓

ピーク発見

↓

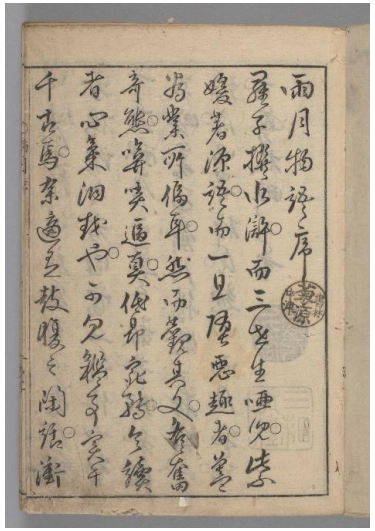
枠座標推定

↓

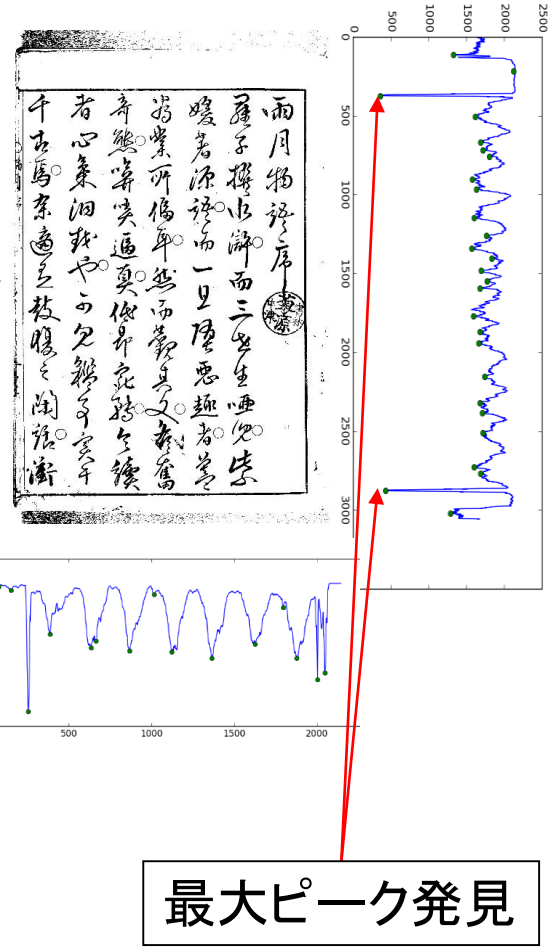
画像カット

↓

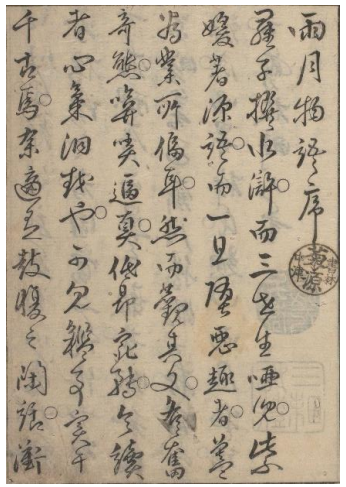
枠を除去した画像



二値化



枠除去



前処理（2/2）：余白除去

原図

↓

画像二値化

↓

ピクセル計算

↓

平均値計算

↓

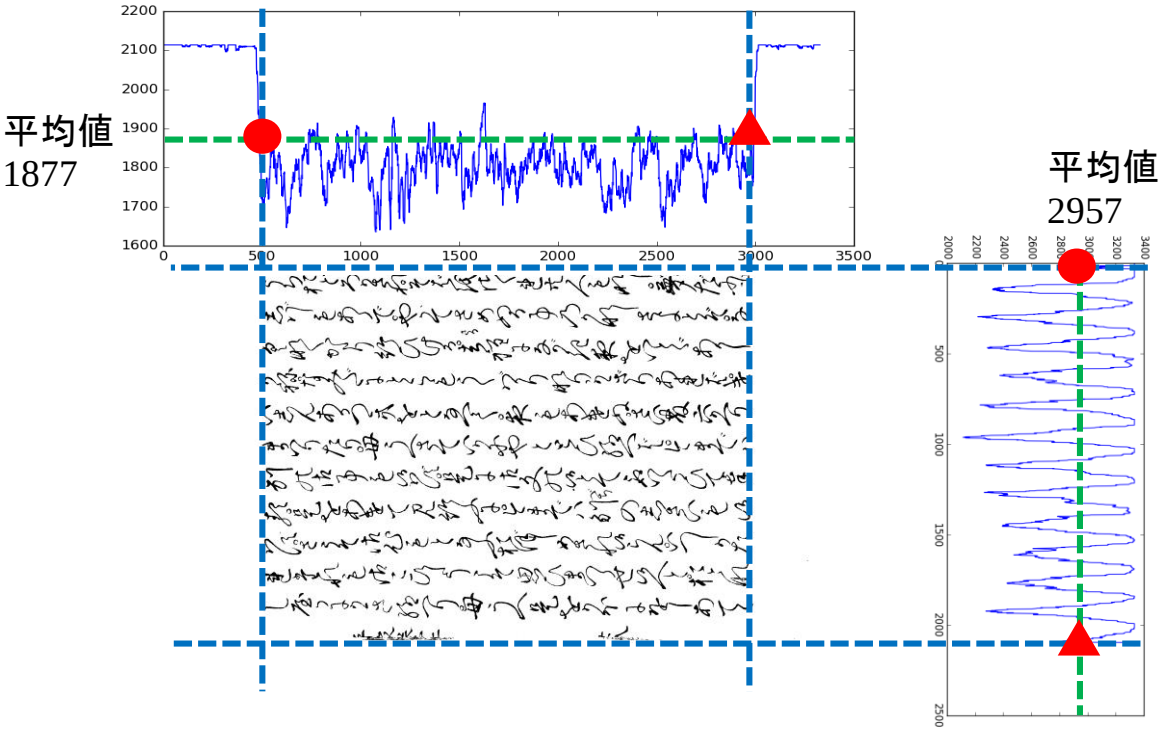
平均値を超えた最初と最後の座標で余白推定

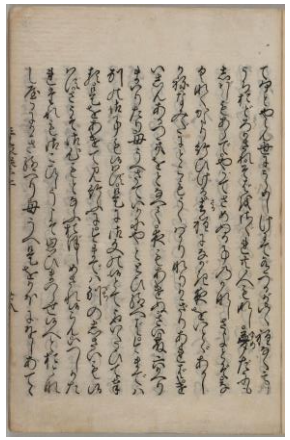
↓

画像カット

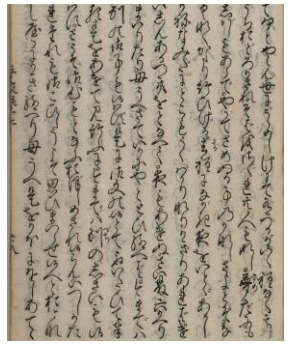
↓

余白を除去した画像



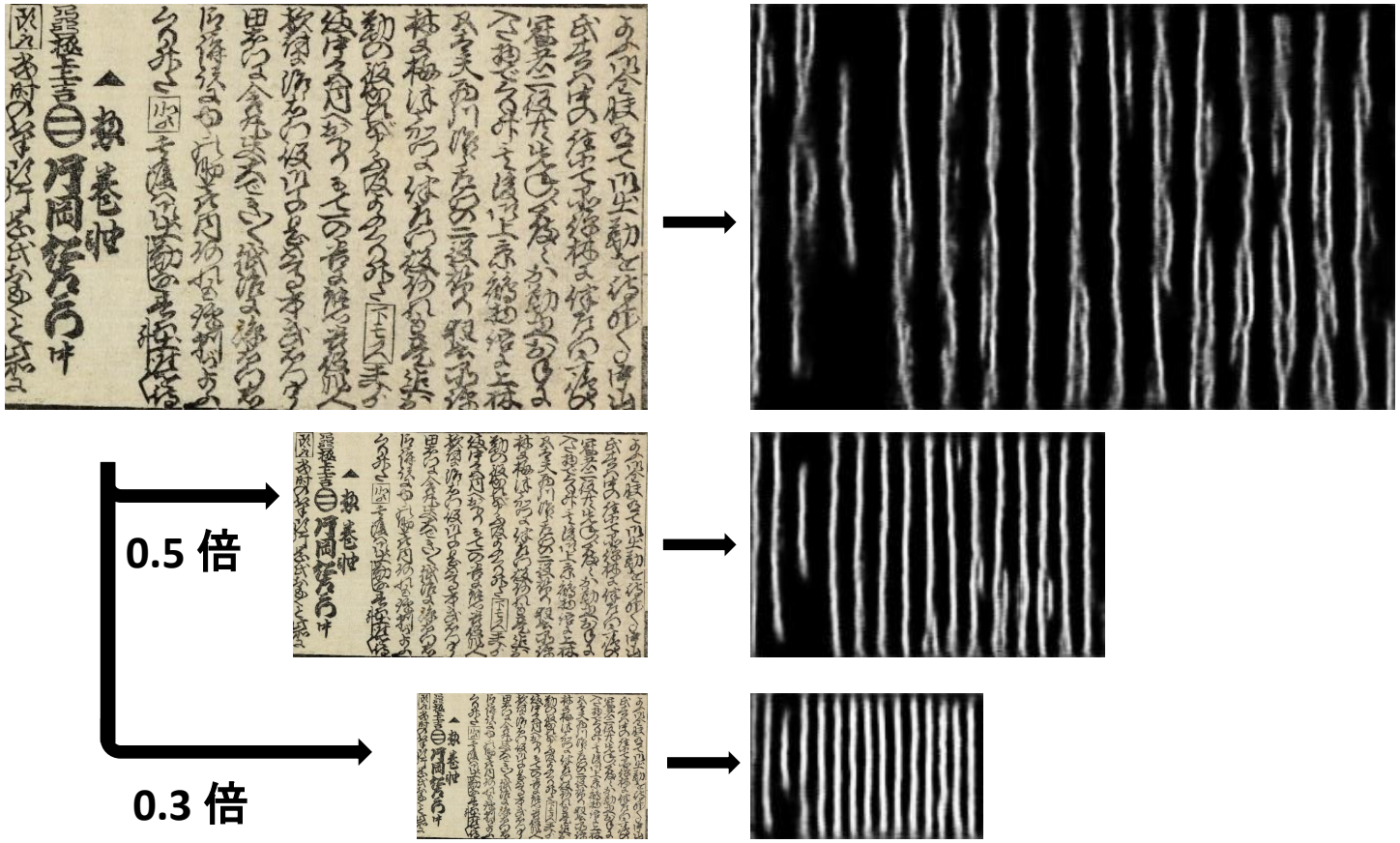


余白除去



テキスト検出(1/2)

- ARU-Netは、歴史的文書の文章行を検出するために使用した深層学習モデル



テキスト検出(2/2)

- 同じ画像をARU-Netで2回処理

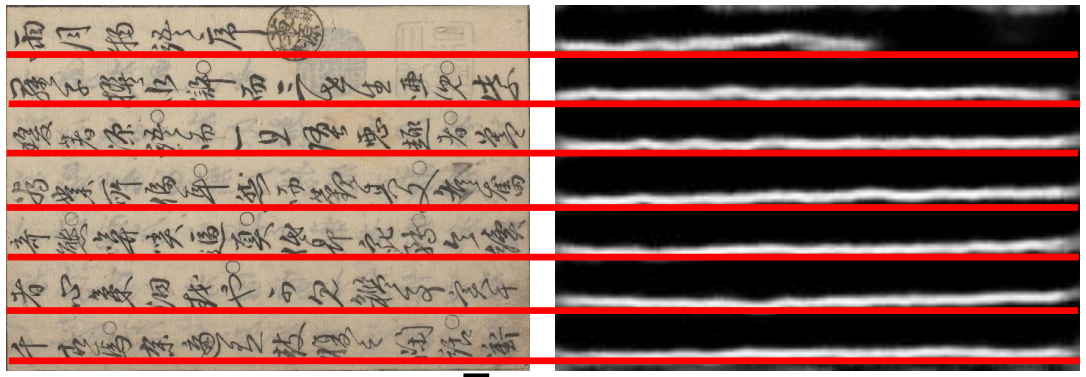


図1

180度回転

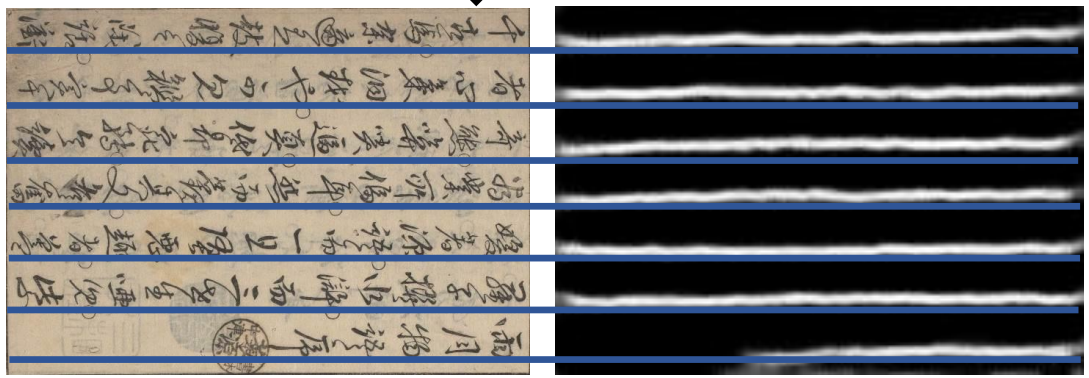


図2

ARU-Netでテキスト検出

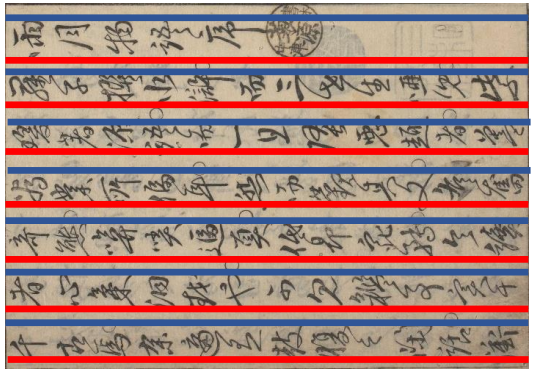
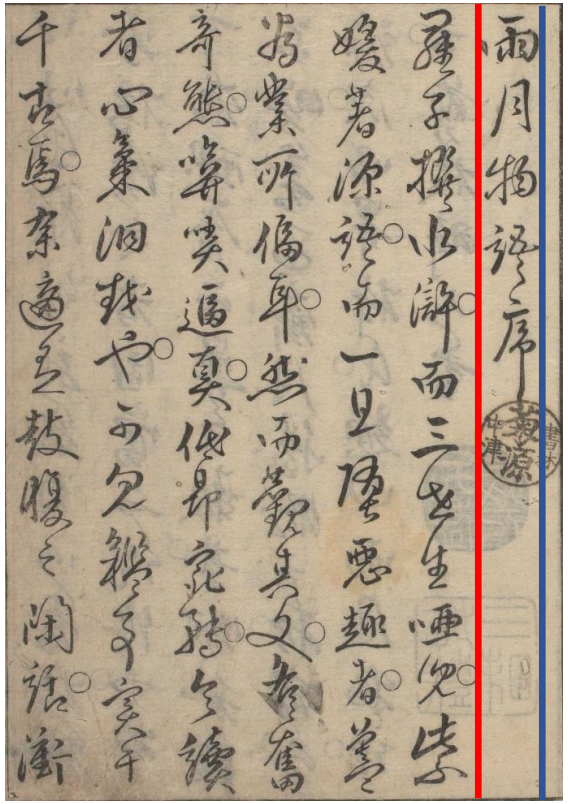


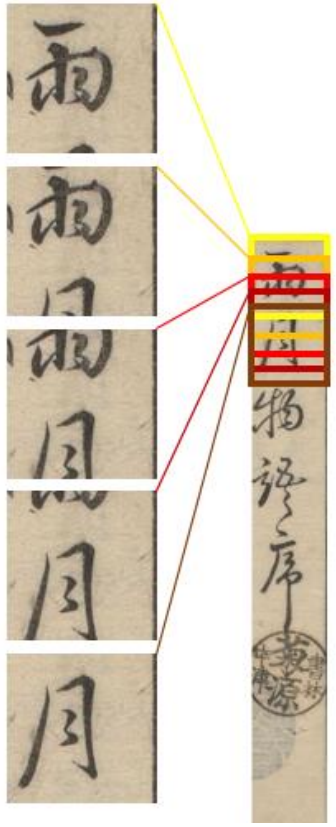
図3　テキスト検出結果

テキスト切り取り

- 赤い線と青い線間のテキストを切り取る
- 上から画像の横の1/5を経て切り出す(正方形)



ARU-Netで検出結果



文字を切り取り

文字認識（AlexNet)

- 切り出した画像を文字認識
- 認識率が最も高い認識結果を選択（赤い長方形）

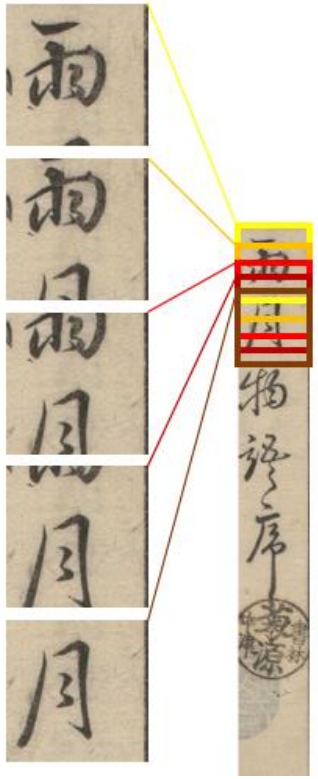
雨90.2% 而8.8% 両1.0%

雨50.4% 両30.2% 四19.4%

四46.2% 目37.5% 日16.3%

月90.5% 目9.0% 口0.5%

月100% 目0.0% 日0.0%



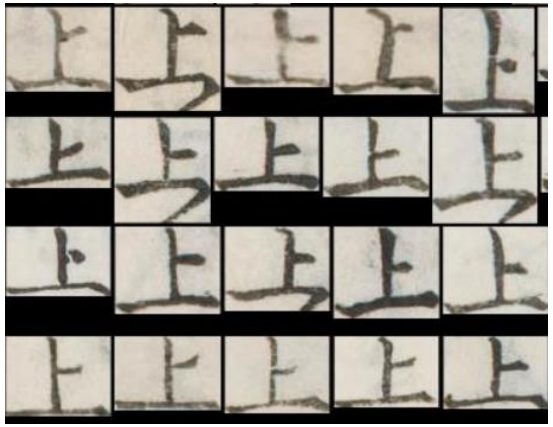
文字認識例

実験条件

- データベース：
  - 人文学オープンデータ共同利用センター（http://codh.rois.ac.jp/）の「浮世風呂」（「うきよぶろ」番号：200015779）（文字種：1817 文字数：60381）
- 実験環境：
  - OS：ubuntu16.04 LTS
  - プログラミング言語：Python

U+306E の 2016	U+3044 い 1514	U+304B か 1503	U+306A な 1336	U+306F は 1330
U+3089 ら 1160	U+307E ま 1079	U+306B に 1073	U+300C 「 1072	U+3068 と 1043
U+3064 つ 952	U+3078 へ 945	U+308B る 912	U+308A り 884	U+3060 だ 869

文字種と文字数（一部）



トレーニング画像例

トレーニング画像増加

- トレーニングデータが不足しているため、データセットを増加
  - 1枚のトレーニング画像の文字を上、下、左、右計4方向に、それぞれ25と50ピクセル移動
    （原図 + 4方向*2パターン = 9枚に拡張）

原図　右へ　左へ　上へ　下へ

文字移動例(25ピクセル)

実験結果

- 認識率が一番高い結果から、認識率が90%以上の結果のみを選出(赤いと青い長方形)
- 一部の文字は正しく認識できたが、一部は正しく認識できなかった

一番高い認識率

U+4E00/0_11	月	100.0%	風	0.0%	左	0.0%
U+4E00/0_12	月	99.93%	風	0.06%	左	0.01%
U+4E00/0_13	風	46.34%	月	40.94%	ほ	4.13%
U+4E00/0_14	ほ	66.56%	ぼ	4.78%	侍	4.25%
U+4E00/0_15	翁	73.23%	海	6.88%	家	5.61%
U+4E00/0_16	海	96.07%	雨	1.04%	歸	0.87%
U+4E00/0_17	海	35.75%	佛	18.18%	朝	9.09%
U+4E00/0_18	野	54.75%	佛	33.65%	朝	3.72%
U+4E00/0_19	朝	67.63%	雲	14.1%	野	9.8%
U+4E00/0_2	一	59.7%	三	26.1%	下	9.93%
U+4E00/0_20	義	68.46%	雨	8.57%	翁	5.62%
U+4E00/0_21	佛	70.25%	義	10.53%	花	5.23%
U+4E00/0_22	語	93.51%	此	2.76%	佛	2.46%
U+4E00/0_23	ぞ	86.93%	聞	12.84%	雲	0.2%
U+4E00/0_24	法	49.98%	金	33.23%	ゾ	5.67%
U+4E00/0_25	え	65.41%	金	6.5%	ほ	4.96%
U+4E00/0_26	義	49.27%	ほ	41.75%	身	2.73%
U+4E00/0_27	事	93.72%	命	4.37%	義	0.6%
U+4E00/0_28	命	97.72%	佛	1.9%	神	0.18%
U+4E00/0_29	ぼ	53.21%	今	8.32%	命	6.67%

文字認識結果

まとめと今後の課題

- ARU-NetとAlexNetに基づく文字認識システムを構築
- より高い精度を達成するために、枠除去と余白除去の手法を提案

- 今後の課題
  - 注釈文字の認識（右の図の赤い枠部分）
  - 画像処理で文字認識率を向上させる