

1. 画像処理の基礎

松田 征也

2020/04/17

0 python+OpenCV のプログラミング環境構築

課題において anaconda を使用して Python 環境の構築をあらかじめ行った。

Anaconda の仮想環境に OpenCV の環境を構築するため、以下の conda コマンドを入力した。

```
conda install -c menpo opencv
```

統合開発プラットフォームは Jupyter Notebook を使用した。実験環境は以下に示す。

- MacOS Catalina バージョン 10.15.4
- Anaconda3 20.02 Python3.7
- numpy 1.18.1
- opencv 3.4.2
- matplotlib 3.1.3

1 numpy を使った行列の四則演算

行列 A , B およびスカラー値 k を定義して、以下の 5 つの演算を試みる。

- 行列の和
- 行列の差
- 行列の積
- スカラー積
- アダマール積

今回, A , B , c の値を以下の式 (1) のように定める.

$$A = \begin{pmatrix} 2.0 & 1.0 \\ 1.5 & 1.3 \end{pmatrix}, B = \begin{pmatrix} 4.0 & 3.0 \\ 8.0 & 1.0 \end{pmatrix}, c = 2 \quad (1)$$

それぞれの 5 つの演算の結果は以下の式 (2) から式 (6) のようになる

- 行列の和

$$A + B = \begin{pmatrix} 6.0 & 4.0 \\ 9.5 & 2.3 \end{pmatrix} \quad (2)$$

- 行列の差

$$A - B = \begin{pmatrix} -2.0 & -2.0 \\ -6.5 & 0.3 \end{pmatrix} \quad (3)$$

- 行列の積

$$AB = \begin{pmatrix} 8.0 & 3.0 \\ 12.0 & 1.3 \end{pmatrix} \quad (4)$$

- スカラー積

$$cA = \begin{pmatrix} 4.0 & 2.0 \\ 3.0 & 2.6 \end{pmatrix} \quad (5)$$

- アダマール積

$$A \otimes B = \begin{pmatrix} 16.0 & 7.0 \\ 16.4 & 5.8 \end{pmatrix} \quad (6)$$

1.1 ソースコード

```

1 import numpy as np
2 A=np.array([[2.0,1.0],[1.5,1.3]])
3 B=np.array([[4.0,3.0],[8.0,1.0]])
4 c=2
5 print("A+B=\n{}".format(A+B))
6 print("A-B=\n{}".format(A-B))
7 print("A*B=\n{}".format(A*B))
8 print("A*c=\n{}".format(A*c))
9 print("A@B=\n{}".format(A@B))

```

1.2 実行結果

```

A+B=
[[6.  4. ]
 [9.5 2.3]]
A-B=
[[-2. -2. ]
 [-6.5 0.3]]
A*B=
[[ 8.   3. ]
 [16.  5.8]]
A*c=
[[ 4.   2. ]
 [3.   2.6]]
A@B=
[[16.  7. ]
 [16.4 5.8]]

```

```
[12.    1.3]]  
A*c=  
[[4.   2. ]  
 [3.   2.6]]  
A@B=  
[[16.   7. ]  
 [16.4  5.8]]
```

2 画像の表示，縮小拡大，回転，二値化

OpenCV を用いて画像の表示，拡大，縮小，回転，二値化を行った。画像は図 1 を使用した。



Figure 1: 画像処理に用いる画像

2.1 ソースコード

```
1 import cv2
2 from IPython.display import Image, display
3 ##画像の表示
4 def imshow(img):
5     '''画像を Notebook 上にインラインで表示する。
6     '''
7     img = cv2.imencode('.png', img)[1]
8     display(Image(img))
```

```

9 | img = cv2.imread('IMG_7562.jpg')
10| imshow(img)
11| img.shape[:2]
12| ##画像の拡大
13| height,width=img.shape[:2]
14| img_double=cv2.resize(img,(width*2,height*2))
15| imshow(img_double)
16| img_double.shape[:2]
17| ##画像の縮小
18| height,width=img.shape[:2]
19| img_half=cv2.resize(img,(int(width/2),int(height/2)))
20| imshow(img_half)
21| img_half.shape[:2]
22| ##画像の回転
23| ##回転の中心を定義
24| center=(int(width/2),int(height/2))
25| ##回転角を指定
26| angle = 90.0
27| ##スケールを指定
28| scale = 1.0
29| #getRotationMatrix2D 関数を使用
30| trans = cv2.getRotationMatrix2D(center,angle,scale)
31| ##アフィン変換
32| image_ro = cv2.warpAffine(img, trans, (width,height))
33| imshow(image_ro)
34| image_ro.shape[:2]
35| ##画像の二値化
36| img_gray=cv2.imread("IMG_7562.JPG",0)
37|
38| # 閾値の設定
39| threshold = 100
40|
41| # 二値化(閾値 100を超えた画素を 255にする。)
42| ret, img_thresh = cv2.threshold(img_gray, threshold, 255, cv2.
   THRESH_BINARY)
43|
44| # 二値化画像の表示
45| imshow(img_thresh)

```

2.2 実行結果

ソースコード中で作成した関数 imshow() により表示させた 4 つの画像を示す。



Figure 2: 二倍拡大画像

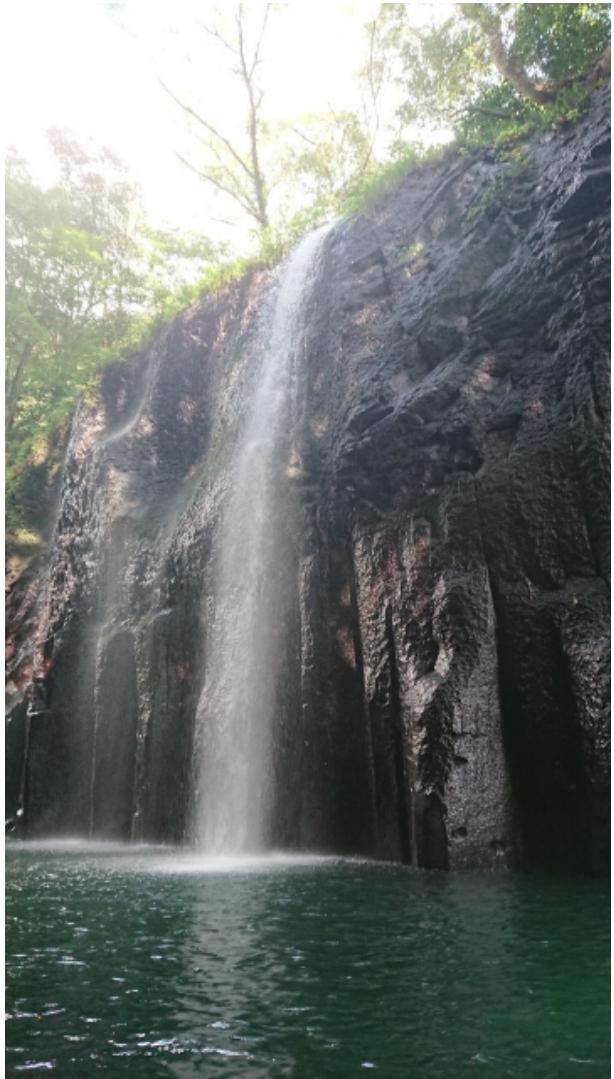


Figure 3: 二分の一縮小画像



Figure 4: 反時計回りに 90 度回転した画像



Figure 5: 二値化した画像

3 2枚の異なる画像の差分画像作成

2枚の異なる画像から差分画像を算出した。ここでは差分画像は、2つの画像の行列 A, B の差の演算結果から、負の値になった要素を 0 に置換することで算出した。

使用した画像を以下の図 6 と図 7 に示す。



Figure 6: 画像 1

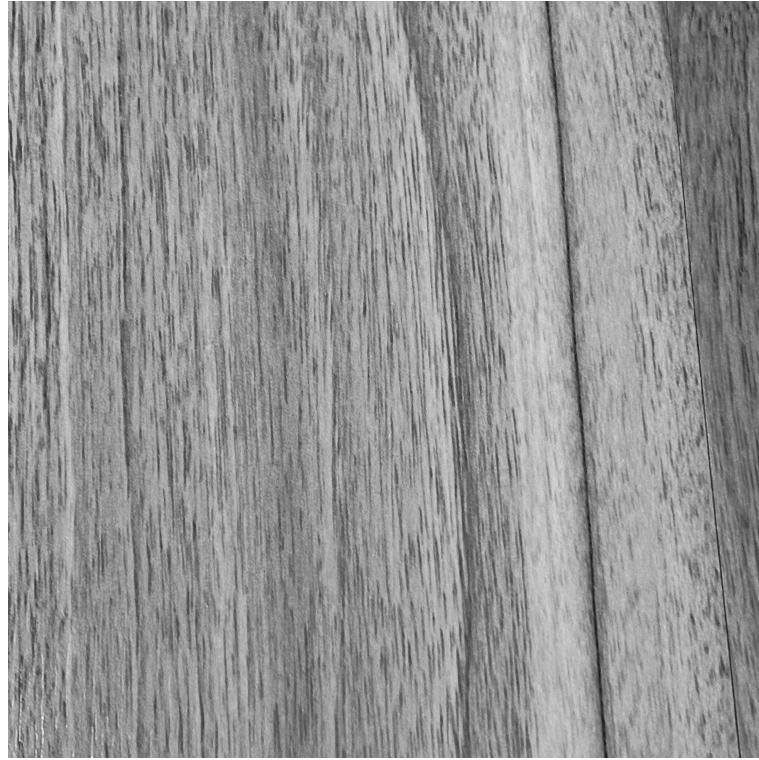


Figure 7: 画像 2

3.1 ソースコード

```
1 # 「背景」となる画像の取り込み（グレースケール）
2 img_src01 = cv2.imread("sample01.jpg", 0)
3 imshow(img_src01)
4 # 「差分」をもった画像の取り込み（グレースケール）
5 img_src02 = cv2.imread("sample02.jpg", 0)
6 imshow(img_src02)
7 # 「背景差分」計算用オブジェクトの作成
8 bgObj = cv2.createBackgroundSubtractorMOG2()
9
10 # 差分となっている「前景領域」に対してマスクをかける
11 fgmask = bgObj.apply(img_src01)
12 fgmask = bgObj.apply(img_src02)
13
14 # 画面に表示
15 imshow(fgmask)
16
17 # 「差分」画像のファイル名
18 bg_diff_path = "./diff.jpg"
19
20 # 「差分」画像の保存
21 cv2.imwrite(bg_diff_path, fgmask)
```

3.2 出力結果

出力された差分画像を図 8 に示す.

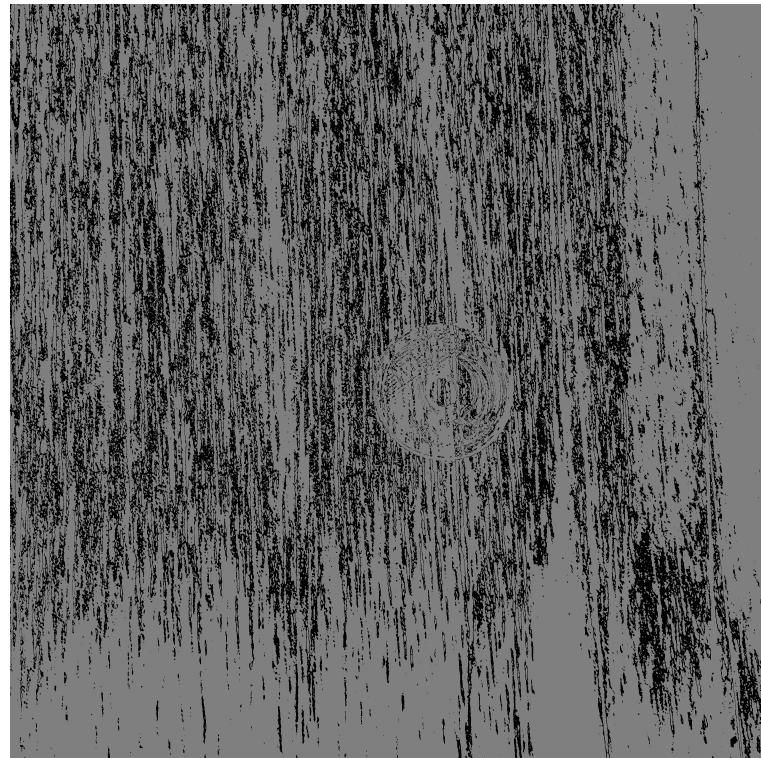


Figure 8: 1 と 2 の差分画像

4 画像の特徴量抽出と図示

4.1 ヒストグラム

画像のヒストグラムとは、画像内の R 値、B 値、G 値の度数分布を表したものである。

Numpy の histogram 関数を使用してヒストグラムを作成し,matplotlib を用いてグラフに表示した。ヒストグラムの作成に使用した画像は図 9 を使用する。



Figure 9: ヒストグラムの作成に使用した画像

4.1.1 ソースコード

```
1 # 入力画像を読み込み
2 img = cv2.imread("IMG_0297.JPG")
3
4 b, g, r = img[:, :, 0], img[:, :, 1], img[:, :, 2]
5
6 # 方法 1(NumPy でヒストグラムの算出)
7 hist_r, bins = np.histogram(r.ravel(), 256, [0, 256])
8 hist_g, bins = np.histogram(g.ravel(), 256, [0, 256])
9 hist_b, bins = np.histogram(b.ravel(), 256, [0, 256])
10
11 # 方法 2(OpenCV でヒストグラムの算出)
12 #hist_r = cv2.calcHist([r], [0], None, [256], [0, 256])
13 #hist_g = cv2.calcHist([g], [0], None, [256], [0, 256])
14 #hist_b = cv2.calcHist([b], [0], None, [256], [0, 256])
15
16 # グラフの作成
17 plt.xlim(0, 255)
18 plt.plot(hist_r, "-r", label="Red")
19 plt.plot(hist_g, "-g", label="Green")
20 plt.plot(hist_b, "-b", label="Blue")
21 plt.xlabel("Pixel_value", fontsize=20)
22 plt.ylabel("Number_of_pixels", fontsize=20)
23 plt.legend()
24 plt.grid()
25 plt.show
```

4.1.2 出力結果

出力結果を図 10 に示す。

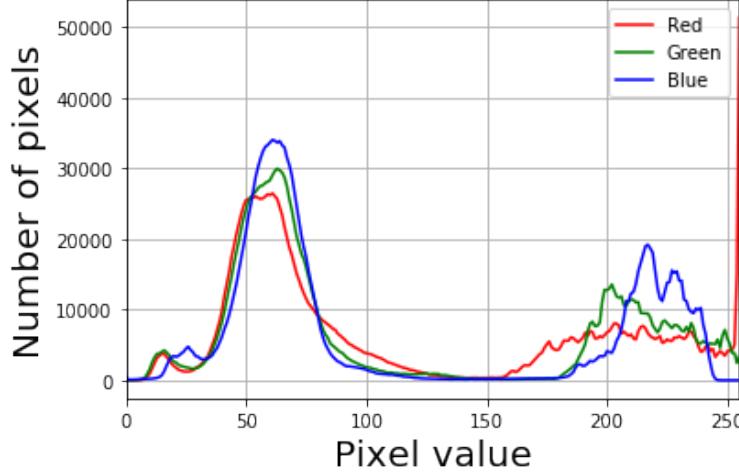


Figure 10: 作成したヒストグラム

4.2 特徴量抽出

二種類の画像（目的画像、全体画像）に対して OpenCV を用いて特徴量抽出を行う。特徴量抽出には SURF のアルゴリズムを使用した。特徴点はキーポイント（画像上の 1 ピクセル）と各キーポイントに対応する特徴ベクトルから構成されている。抽出された特徴量を視覚的表すことを目標とするため、OpenCV の drawKeypoints 関数により特徴ベクトルのノルムの大きさに応じて円を表示させる。

4.2.1 ソースコード

```
1 #一つ目の画像
2 img1= cv2.imread('IMG_8814.JPG')
3 surf = cv2.xfeatures2d.SURF_create(400)
4 kp1 = surf.detect(img1)
5 img1_sift = cv2.drawKeypoints(img1, kp1, None, flags=4)
6 imshow(img1_sift)
7 img_surf1= "./surf1.jpg"
8 cv2.imwrite(img_surf1,img1_sift)
9 #二つ目の画像
10 img2= cv2.imread('IMG_8815.JPG')
11 surf = cv2.xfeatures2d.SURF_create(400)
12 kp2 = surf.detect(img2)
13 img2_sift = cv2.drawKeypoints(img2, kp2, None, flags=4)
14 imshow(img2_sift)
15 img2_surf= "./surf2.jpg"
16 cv2.imwrite(img2_surf,img2_sift)
```

4.3 実行結果

全体画像と目的画像に対し、SURF を用いて特微量抽出を行った結果をそれぞれ図 11, 図 12 に示す。



Figure 11: 目的画像

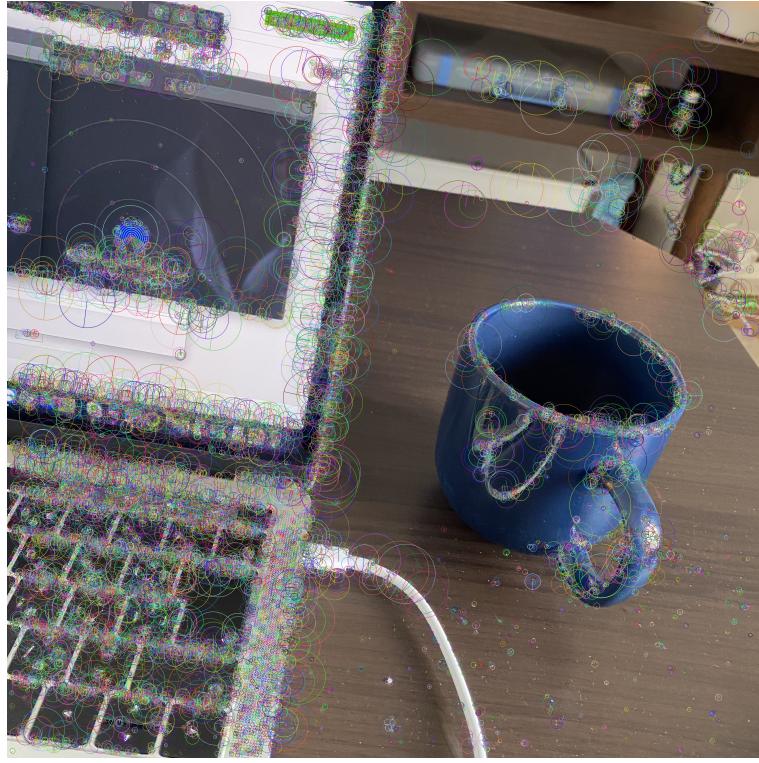


Figure 12: 全体画像

4.4 特徴点のマッチング

次に目的画像であるコップを、全体画像から検出する手法を検討する。

1. 目的画像および全体画像をグレースケール画像に変換する
2. それぞれの画像に対し、SIFT 特微量,SURF 特微量,AKAZE 特微量を抽出する。
3. bf.knnMatch にそれぞれの画像の特微量記述子 (128 次元ベクトル) を渡し距離を計算する。
4. $k = 2$, $ratio = 0.6$ と設定し、des1 のそれぞれの点に対して、最も近い des2 の点 2 つを選び、マッチングインスタンスを出力する。
5. 最も近い点が 2 番目に近い点の距離の 0.6 倍以下の場合採択する。

以上の手順から、最後まで残った目的画像と全体画像の特徴点の組をマッチング結果とする。

4.4.1 ソースコード

```
1 img1 = cv2.imread('IMG_8814.jpg',0)
2 img2 = cv2.imread('IMG_8815.jpg',0)
3 #特徴抽出機の生成 (SIFT,SURF,AKAZE)
4 detector = cv2.xfeatures2d.SIFT_create()
5 #detector = cv2.xfeatures2d.SURF_create()
6 #detector = cv2.AKAZE_create()
7 #kp は特徴的な点の位置 des は特徴を現すベクトル
8 kp1, des1 = detector.detectAndCompute(img1, None)
9 kp2, des2 = detector.detectAndCompute(img2, None)
10 #特徴点の比較機
11 bf = cv2.BFM_matcher()
12 matches = bf.knnMatch(des1,des2, k=2)
13 #割合試験を適用
14 good = []
15 match_param = 0.6
16 for m,n in matches:
17     if m.distance < match_param*n.distance:
18         good.append([m])
19 #cv2.drawMatchesKnn は適合している点を結ぶ画像を生成する
20 img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good, False, flags=10)
21 imshow(img3)
```

4.4.2 出力結果

SIFT による特徴マッチングの結果を以下の図 13, SURF による特徴マッチングの結果を以下の図 14, AKAZE による特徴マッチングの結果を以下の図 15 に示す。

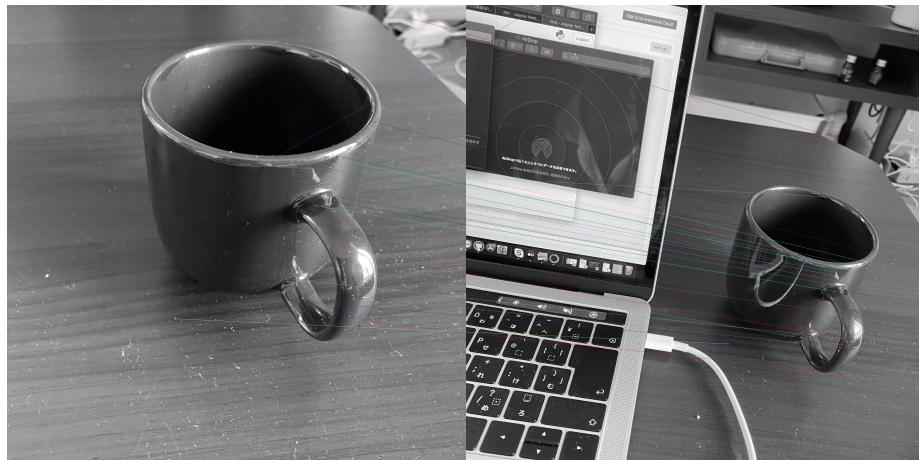


Figure 13: SIFT による特徴マッチングの結果

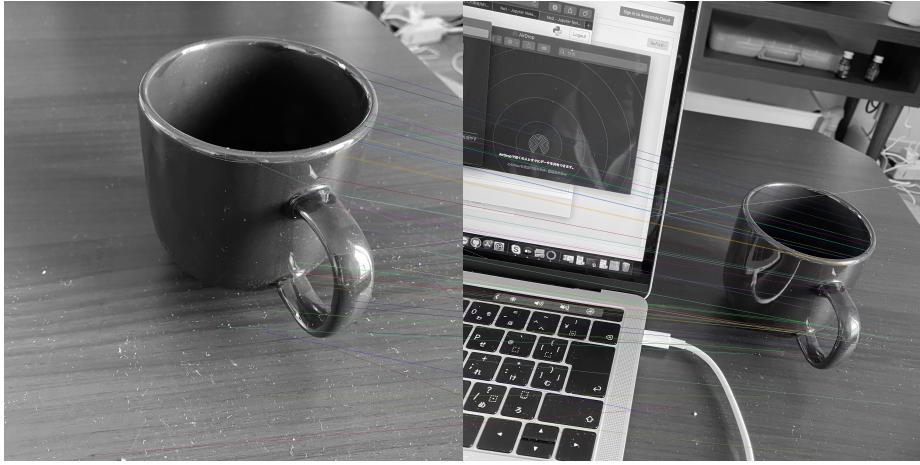


Figure 14: SURF による特徴マッチングの結果

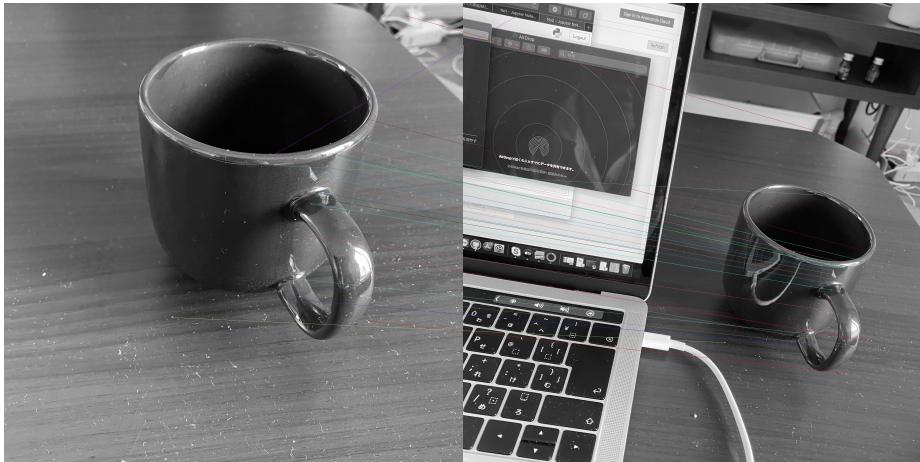


Figure 15: AKAZE による特徴マッチングの結果

どの特徴抽出機を使用した場合も正しく視覚的な検出が行えた。

5 2. 医用画像診断

5.1 二値分類手法の提案

本課題では今後に向けて,CNN(畳み込みニューラルネットワーク)を使用してMRI画像の腫瘍の有無による識別を行うアルゴリズムを作成する。アルゴリズムを作成するにあたり統合開発環境はGoogleが無償で提供しているGoogle colabを使用した。実験環境は以下に示す。

- MacOS Catalina バージョン 10.15.4
- Python3.6
- numpy 1.18.1
- pandas 1.0.3
- matplotlib 3.1.3
- sklearn 0.22.2.post1
- keras 2.3.1

5.2 データセットの準備

与えられたデータセットは訓練データ 8980 枚, 検証データ 1448 枚, テストデータ 2458 枚である。これらのデータセットを Numpy 配列として読み込み, テキストファイルとして保存した。

5.2.1 ソースコード

```

1 import numpy as np
2 from PIL import Image
3 import os
4 import sys
5 def readImg(paths):
6     N = len(paths)
7     # 画像読み込み準備
8     imgs = [[] for i in range(N)]
9     # 正解データ作成
10    imgs_targets = []
11    for k, path in enumerate(paths):
12        #label = 画像が入ってるフォルダ名
13        label = os.path.basename(os.path.dirname(path))
14        if label == "0":
15            imgs_targets.append(0)
16        else:
17            imgs_targets.append(1)
18        imgs[k] = np.asarray(Image.open(path))
19        sys.stderr.write('{}枚目\r'.format(k))
20        sys.stderr.flush()
21        sys.stderr.write('\n')
22    imgs = np.array(imgs, dtype = "float32")
23    imgs_targets = np.array(imgs_targets, dtype = "int32")
24    return imgs, imgs_targets
25 from glob import glob
26 # データセットのあるパス
27 main_path = "/content/drive/MyDrive/ColabNotebooks/datasets/"
28 train_path = main_path + "train/"
29 val_path = main_path + "val/"
30 test_path = main_path + "test/"
31 # 全画像のパス読み込み
32 train_paths = np.array(sorted(glob(train_path + "**/*.png")))

```

```

33     val_paths = np.array(sorted(glob(val_path + "**/*.png")))
34     test_paths = np.array(sorted(glob(test_path + "**/*.png")))
35     print(len(train_paths), len(val_paths), len(test_paths))
36     x_val, y_val = readImg(val_paths)
37     x_test, y_test= readImg(test_paths)
38     x_train,y_train=readImg(train_paths)
39     np.save(main_path + "x_val", x_val)
40     np.save(main_path + "y_val", y_val)
41     np.save(main_path + "x_test", x_test)
42     np.save(main_path + "y_test", y_test)
43     np.save(main_path + "x_train", x_train)
44     np.save(main_path + "y_train", y_train)

```

5.3 データの呼び出し及び前処理

与えられたデータセットの特徴量は0～255の範囲で与えられているため、0～1の範囲に収まるように正規化を行う。また、データセットは3次元テンソルの形状をしているため、4次元テンソルに変換する処理を行った。また、データセットはラベルから順に整列されているためデータセットの順序をランダムに並び替える処理を行った。

5.3.1 ソースコード

```

1 import numpy as np
2 x_val = np.load(main_path + "x_val.npy")
3 y_val = np.load(main_path + "y_val.npy")
4 x_train= np.load(main_path + "x_train.npy")
5 y_train= np.load(main_path + "y_train.npy")
6 x_test= np.load(main_path + "x_test.npy")
7 y_test= np.load(main_path + "y_test.npy")
8 x_train=x_train/255
9 x_val=x_val/255
10 x_test=x_test/255
11 x_train=x_train.reshape(8980,224,224,1)
12 x_val=x_val.reshape(1448,224,224,1)
13 x_test=x_test.reshape(2458,224,224,1)
14 def shuffle(x,y):
15     np.random.seed(1)
16     np.random.shuffle(x)
17     np.random.seed(1)
18     np.random.shuffle(y)
19     shuffle(x_train,y_train)
20     shuffle(x_val,y_val)
21     shuffle(x_test,y_test)

```

5.4 CNN モデルの構築

構築したCNNモデルはConv2D層とMaxPooling 2 D層のスタックである。このCNNモデルはサイズ(224,224,1)の入力を処理するように設定している。また、腫瘍ありか腫瘍なしを判別する2クラス分類であるため、モデルの最終出力

は1ユニットであり活性化関数にはシグモイド関数を用いた。過学習の抑制を行うためにコールバック関数としてEarlyStopping関数を使用している。

5.4.1 ソースコード

```
1 import keras
2 from keras.utils import np_utils
3 from keras.layers.convolutional import Conv2D, MaxPooling2D
4 from keras.models import Sequential
5 from keras.layers.core import Dense, Dropout, Activation, Flatten
6 import numpy as np
7 import pandas as pd
8 from sklearn.model_selection import train_test_split
9 import matplotlib.pyplot as plt
10 model = Sequential()
11
12 model.add(Conv2D(32, (3, 3), padding='same',
13                  input_shape=(224,224,1)))
14 model.add(Activation('relu'))
15 model.add(Conv2D(32, (3, 3)))
16 model.add(Activation('relu'))
17 model.add(MaxPooling2D(pool_size=(2, 2)))
18 model.add(Dropout(0.25))
19 model.add(Conv2D(64, (3, 3), padding='same'))
20 model.add(Activation('relu'))
21 model.add(Conv2D(64, (3, 3)))
22 model.add(Activation('relu'))
23 model.add(MaxPooling2D(pool_size=(2, 2)))
24 model.add(Dropout(0.25))
25 model.add(Flatten())
26 model.add(Dense(512))
27 model.add(Activation('relu'))
28 model.add(Dropout(0.5))
29 model.add(Dense(1))
30 model.add(Activation('sigmoid'))
31 model.compile(loss='binary_crossentropy',
32                 optimizer="rmsprop",
33                 metrics=['accuracy'])
34 es_cb = keras.callbacks.EarlyStopping(monitor='val_loss', patience=0,
35                                       verbose=0, mode='auto')
36 tb_cb = keras.callbacks.TensorBoard(log_dir=".//logs", histogram_freq
37                                     =1)
38 history = model.fit(x_train, y_train, batch_size=128, epochs=5,
39                      validation_data = (x_val, y_val),verbose=1,callbacks=[es_cb, tb_cb])
40 model.save(main_path+"kadai2.h5")
```

5.4.2 実行結果

```
1 Using TensorFlow backend.
2 Train on 8980 samples, validate on 1448 samples
3 Epoch 1/5
4 8980/8980 [=====] - 53s 6ms/step - loss: 0.6663 -
   accuracy: 0.7389 - val_loss: 0.5640 - val_accuracy: 0.7762
```

```

5 | Epoch 2/5
6 | 8980/8980 [=====] - 49s 5ms/step - loss: 0.4012 -
|   accuracy: 0.8297 - val_loss: 0.4952 - val_accuracy: 0.7762
7 | Epoch 3/5
8 | 8980/8980 [=====] - 49s 5ms/step - loss: 0.2998 -
|   accuracy: 0.8867 - val_loss: 0.4067 - val_accuracy: 0.8391
9 | Epoch 4/5
10 | 8980/8980 [=====] - 49s 6ms/step - loss: 0.1903 -
|   accuracy: 0.9322 - val_loss: 0.5079 - val_accuracy: 0.8225

```

5.4.3 構築した CNN モデルのアーキテクチャ

```

1 Model: "sequential_1"
2 -----
3 Layer (type) Output Shape Param #
4 -----
5 conv2d_1 (Conv2D) (None, 224, 224, 32) 320
6 -----
7 activation_1 (Activation) (None, 224, 224, 32) 0
8 -----
9 conv2d_2 (Conv2D) (None, 222, 222, 32) 9248
10 -----
11 activation_2 (Activation) (None, 222, 222, 32) 0
12 -----
13 max_pooling2d_1 (MaxPooling2D) (None, 111, 111, 32) 0
14 -----
15 dropout_1 (Dropout) (None, 111, 111, 32) 0
16 -----
17 conv2d_3 (Conv2D) (None, 111, 111, 64) 18496
18 -----
19 activation_3 (Activation) (None, 111, 111, 64) 0
20 -----
21 conv2d_4 (Conv2D) (None, 109, 109, 64) 36928
22 -----
23 activation_4 (Activation) (None, 109, 109, 64) 0
24 -----
25 max_pooling2d_2 (MaxPooling2D) (None, 54, 54, 64) 0
26 -----
27 dropout_2 (Dropout) (None, 54, 54, 64) 0
28 -----
29 flatten_1 (Flatten) (None, 186624) 0
30 -----
31 dense_1 (Dense) (None, 512) 95552000
32 -----
33 activation_5 (Activation) (None, 512) 0
34 -----
35 dropout_3 (Dropout) (None, 512) 0
36 -----
37 dense_2 (Dense) (None, 1) 513
38 -----
39 activation_6 (Activation) (None, 1) 0
40 -----
41 Total params: 95,617,505
42 Trainable params: 95,617,505
43 Non-trainable params: 0
44 -----

```

5.5 テストデータによる評価結果

作成したモデルをテストデータを用いて評価する。

5.5.1 ソースコード

```
1 #作成済みのモデルを呼び出す
2 import keras
3 model_new=keras.models.load_model("/content/drive/My_Drive/Colab_
Notebooks/datasets/kadai2.h5")
4 test_loss,test_acc=model_new.evaluate(x_test,y_test)
5 test_acc
```

5.5.2 実行結果

```
1 2458/2458 [=====] - 5s 2ms/step
2 0.8852725625038147
```

テストデータでの認識率は約 89 %となった。VGG16 を使用した場合の認識率(85 %～90 %)と比べて妥当な認識率と考えられる。

5.6 混同行列の出力及び画像の傾向解析

5.6.1 ソースコード

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sn
4 from sklearn.metrics import confusion_matrix
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from sklearn.metrics import confusion_matrix
8 #テストデータに対する混同行列の出力
9 predict_classes = model_new.predict_classes(x_test)
10 true_classes = y_test
11 def print_cmx(y_true, y_pred):
12     labels = sorted(list(set(y_true)))
13     cmx_data = confusion_matrix(y_true, y_pred, labels=labels)
14
15     df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)
16
17     plt.figure(figsize = (10,10))
18     sn.heatmap(df_cmx, annot=True, fmt='g' ,square = True)
19     plt.show()
20
21 print_cmx(true_classes, predict_classes)
22 cm = confusion_matrix(true_classes,predict_classes)
23 #予測結果とラベルが異なる画像の保存
24 import gc, keras, os
```

```
25 import numpy as np
26 from keras.models import load_model
27 from keras.datasets import mnist
28 from PIL import Image, ImageDraw
29 pred = model_new.predict_classes(x_test)# Write error images
30 for i, (p, y) in enumerate(zip(pred, y_test)):
31     if p != y:
32         image = x_test[i]*255
33         img = Image.fromarray(image.reshape((image.shape[0], image.
34             shape[1])).astype(np.uint8))
35         print(img)
36         img.save(main_path+"result_img/"+ "{0}-c{1}-p{2}.png".format(i,
37             y, p)) # c=correct, p=predi
```

5.6.2 出力されたテストデータに対する可視化された混同行列

モデルをテストデータで評価した結果に対する混同行列を図 16 に示す。

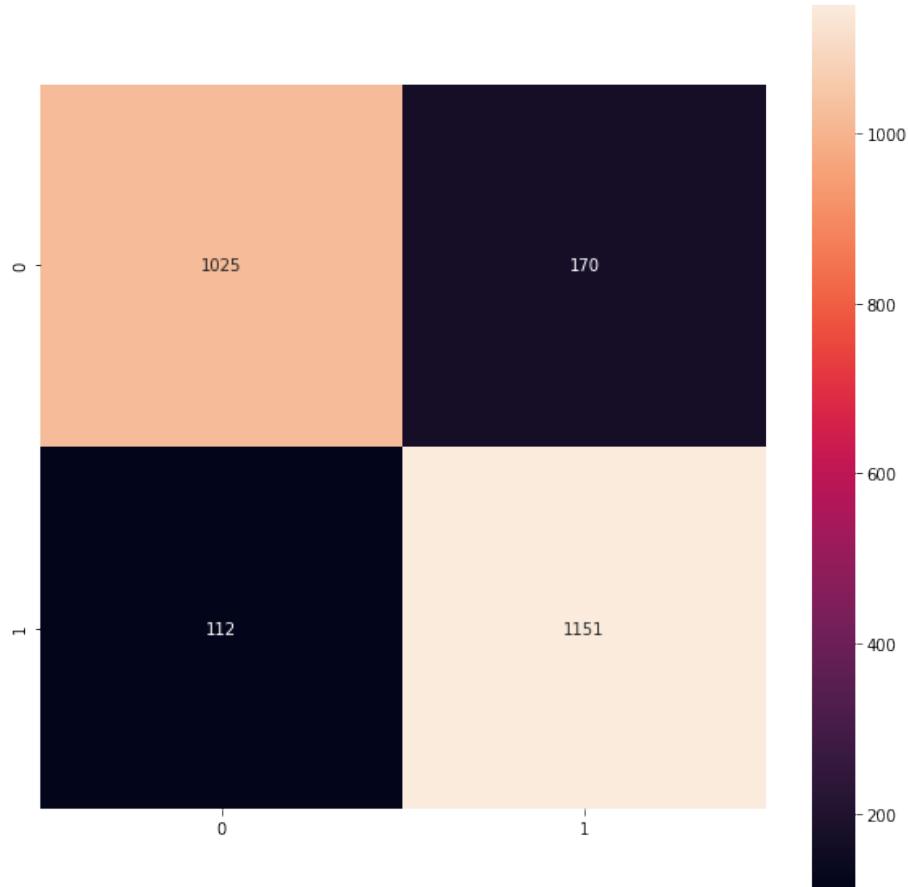


Figure 16: テストデータに対する可視化された混同行列

5.6.3 考察

混同行列より腫瘍ありを陽性とし、二値分類問題における評価指標として正解率、適合率、再現率、F 値を計算すると以下の図 1 のようになった。

Table 1: 二値分類問題における評価指標

正答率	0.885
適合率	0.911
再現率	0.871
F 値	0.891

この問題において実際に腫瘍が存在するにも関わらず腫瘍が存在しないと判断する可能性を減らすことが重要となる。そのため、再現率をあげるようなモ

ルに改良を行っていく必要があると考えられる。実際に腫瘍ありの画像に対し、腫瘍なしと判断した画像のうち 1 枚を以下の図 17 に示す。また、実際に腫瘍ありの画像に対し正しく腫瘍ありと判断できた画像のうち 1 枚を図 18 に示す。

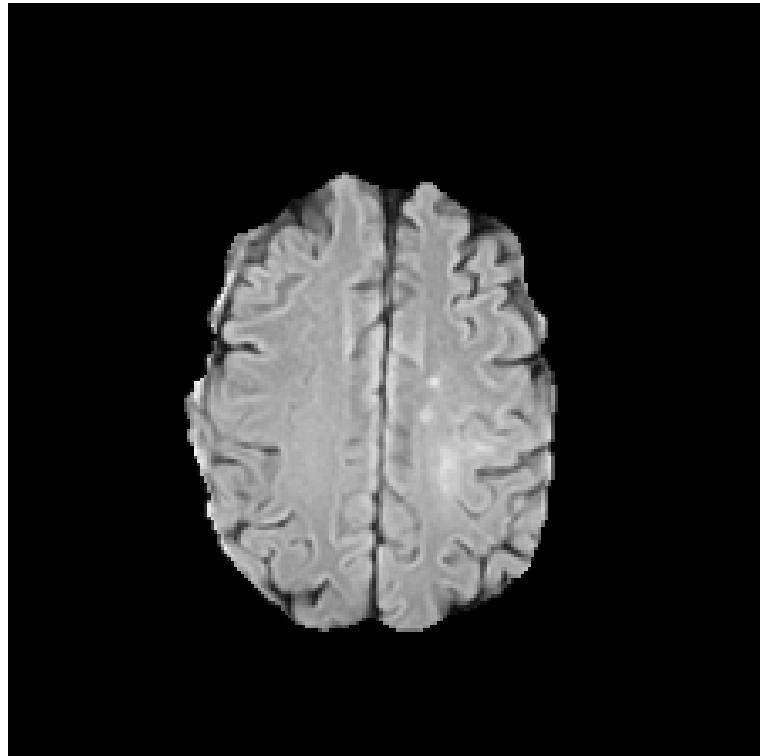


Figure 17: 誤診断された腫瘍あり画像

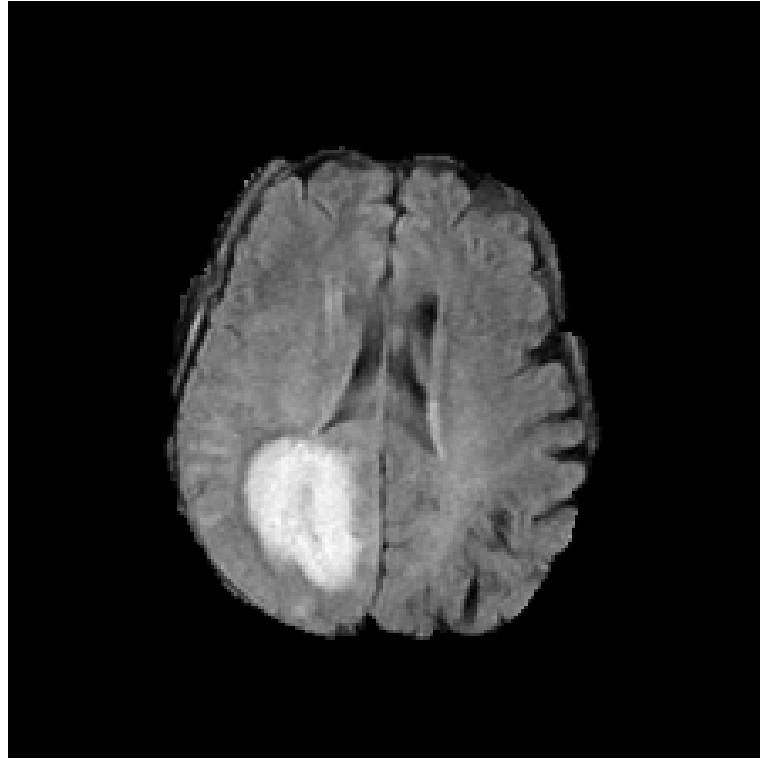


Figure 18: 正しく診断された腫瘍あり画像

腫瘍とされる部分は画像の左右を見比べて、白く変化している部分であると考えられる。腫瘍が存在するにも関わらず存在しないと判断された図 17 では腫瘍である部分が白く変化している部分を人の目で見ても認識する事は困難である。これに比べ、正しく診断された図 18 では左右を見比べて明らかに白く変化している部分が認識できる。他の画像でも同様の傾向が見られた。よって腫瘍に対し腫瘍でないと誤認識している画像では人の目で見てもあまり認識できないほどの画像であることが考えられる。