きれいな文字を書くためには

徳島県立脇町高等学校　　３年
町田若菜
高橋美優
住吉由莉佳

01　研究の目的

1　どのように書いたら文字が綺麗に見えるのか。

2　普段文字を書く機会が多いため文字をきれいに書きたい。

文字のきれいさを表し、フィードバックする

02　きれいな文字の定義について

きれいな文字とは主観で判断される
しかし、客観的にみて綺麗かどうか判断したい

▼

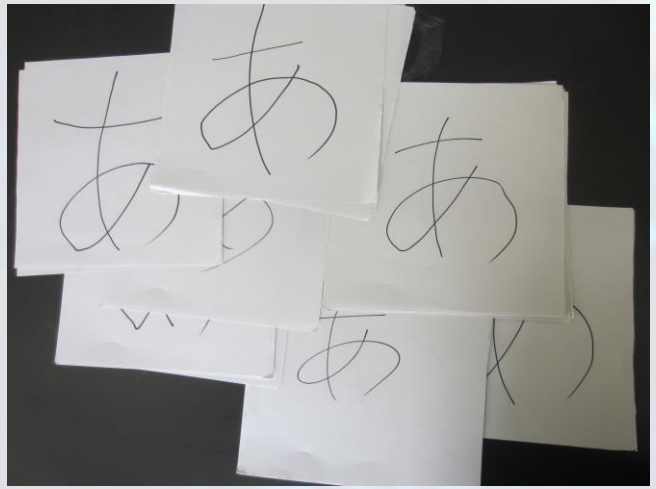
お手本設定には主観を用いるが、比較ではプログラムの客観性を生かすことができる

03　研究の方法

04　大まかな流れ

1

サンプルの収集
（７枚）



2

プログラムを作成
（四分割と絶対値
差分利用）

3

人にフィードバック
（再度数値化）

05　実際の流れ

1. Google Colaboratory の環境でPython言語でプログラムを作る(教師あり学習)
2. 出力結果を人にフィードバックする

↓

実際に文字がきれいになったのか確認する

06　失敗例

1. **特徴点マッチング**
→得られる特徴点が少なく、極端に高い値や低い値が出力され、きれいさを測ることができなかった

2. **座標上での積関数の利用**
→文字の大きさによってトリミングが異なり、比較自体がうまくいかなかった

↓

今の絶対値差分利用のコードに決定

07　実行したコード

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

def upload_images():
    uploaded = files.upload()
    images = []
    for name, data in uploaded.items():
        image = cv2.imdecode(np.frombuffer(data, np.uint8), cv2.IMREAD_COLOR)
        images.append((name, image))
    return images

def preprocess_image(image):
    # ノイズ除去
    denoised_img = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    # グレースケール化
    gray = cv2.cvtColor(denoised_img, cv2.COLOR_BGR2GRAY)
    # アダプティブ閾値処理
    adaptive_thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    # 小さなノイズの除去（形態学的処理）
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(adaptive_thresh, cv2.MORPH_OPEN, kernel, iterations=2)

    # 連結成分解析によるノイズ除去
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(opening, connectivity=8)
    min_size = 150  # 最小の連結成分のサイズを設定
    cleaned_img = np.zeros(opening.shape, dtype=np.uint8)
    for i in range(1, num_labels):  # 0は背景ラベルなのでスキップ
        if stats[i, cv2.CC_STAT_AREA] >= min_size:
            cleaned_img[labels == i] = 255

    # リサイズ
    resized_img = cv2.resize(cleaned_img, (800, 800))
    return resized_img

def find_keypoints_and_descriptors(image):
    orb = cv2.ORB_create()
    keypoints, descriptors = orb.detectAndCompute(image, None)
    return keypoints, descriptors
```

←画像のインポート、アップロード

←前処理関数

←ORB特徴点の検出

```python
def match_keypoints(descriptors1, descriptors2):
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(descriptors1, descriptors2)S
    matches = sorted(matches, key=lambda x: x.distance)
    return matches

def calculate_similarity_absolute_difference(matches, max_distance=100):
    if len(matches) == 0:
        return 0  # マッチがない場合、最低のスコアを返す
    total_distance = sum([m.distance for m in matches])
    avg_distance = total_distance / len(matches)
    similarity_score = max(0, 100 - (avg_distance / max_distance) * 100)
    return similarity_score

def draw_center_lines(image):
    if len(image.shape) == 2:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    height, width = image.shape[:2]
    cv2.line(image, (width // 2, 0), (width // 2, height), (0, 255, 0), 2)
    cv2.line(image, (0, height // 2), (width, height // 2), (0, 255, 0), 2)
```

←BFMatcherオブジェクトの作成と距離に基づいての評価をする

←いいマッチを抽出後全体のマッチ数で割り、類似度計算をする

←四分割のラインを書く

```python
def visualize_comparison(reference_image, target_image):
    # Make the red channel of the reference image maximum
    reference_image_color = cv2.cvtColor(reference_image, cv2.COLOR_GRAY2BGR)
    reference_image_color[:, :, 2] = 255

    target_image_color = cv2.cvtColor(target_image, cv2.COLOR_GRAY2BGR)
    combined_image = cv2.addWeighted(reference_image_color, 0.5, target_image_color, 0.5, 0)
    combined_image_with_lines = draw_center_lines(combined_image)
    plt.imshow(cv2.cvtColor(combined_image_with_lines, cv2.COLOR_BGR2RGB))
    plt.show()
```

←お手本画像を赤色に変換
　お手本と比較画像を加重平均によって重ね、四分割のラインを入れる

←合成画像の表示

```python
def build_database(images):
    database = []
    for name, image in images:
        processed_image = preprocess_image(image)
        keypoints, descriptors = find_keypoints_and_descriptors(processed_image)
        database.append((name, processed_image, keypoints, descriptors))
    return database

def compare_with_database(reference_image_processed, database):
    keypoints1, descriptors1 = find_keypoints_and_descriptors(reference_image_processed)
    for name, processed_image, keypoints2, descriptors2 in database:
        matches = match_keypoints(descriptors1, descriptors2)
        similarity = calculate_similarity_absolute_difference(matches)
        print(f"画像: {name}, 類似度: {similarity:.2f}")
        visualize_comparison(reference_image_processed, processed_image)

# 画像のアップロードと前処理
print("お手本画像をアップロードしてください。")
ref_images = upload_images()
ref_name, ref_image = ref_images[0]  # 最初の画像をお手本画像として使用
ref_image_processed = preprocess_image(ref_image)

# 比較する画像データベースの構築
print("比較する画像をアップロードしてください。")
uploaded_images = upload_images()
database = build_database(uploaded_images)

# データベース内の画像との比較
compare_with_database(ref_image_processed, database)
```

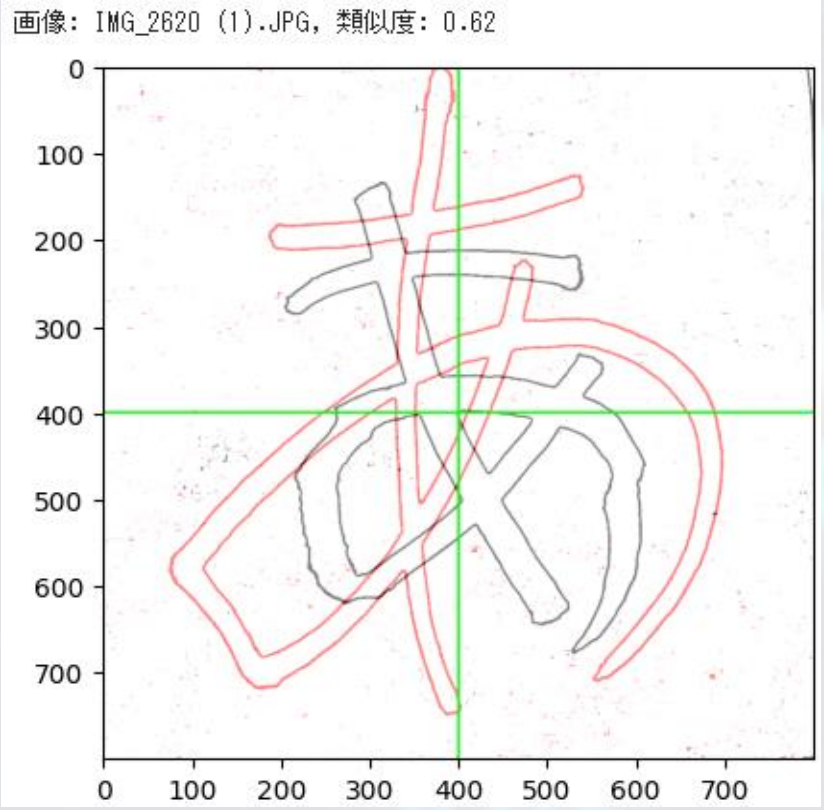
←データベース作成(比較したい文字)

←お手本画像とデータベースの比較

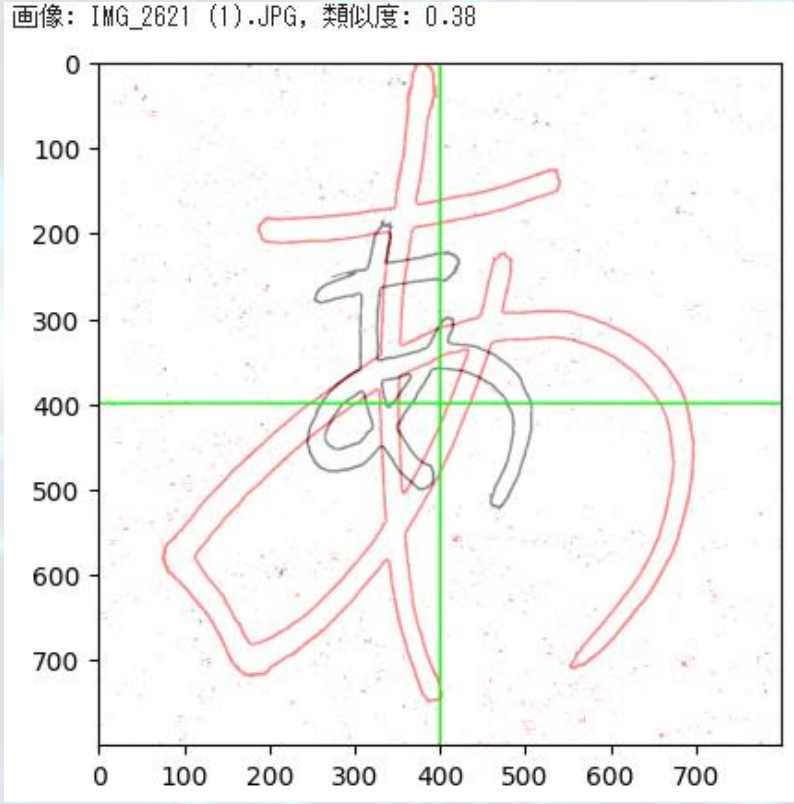
←実行コード

08 実験結果

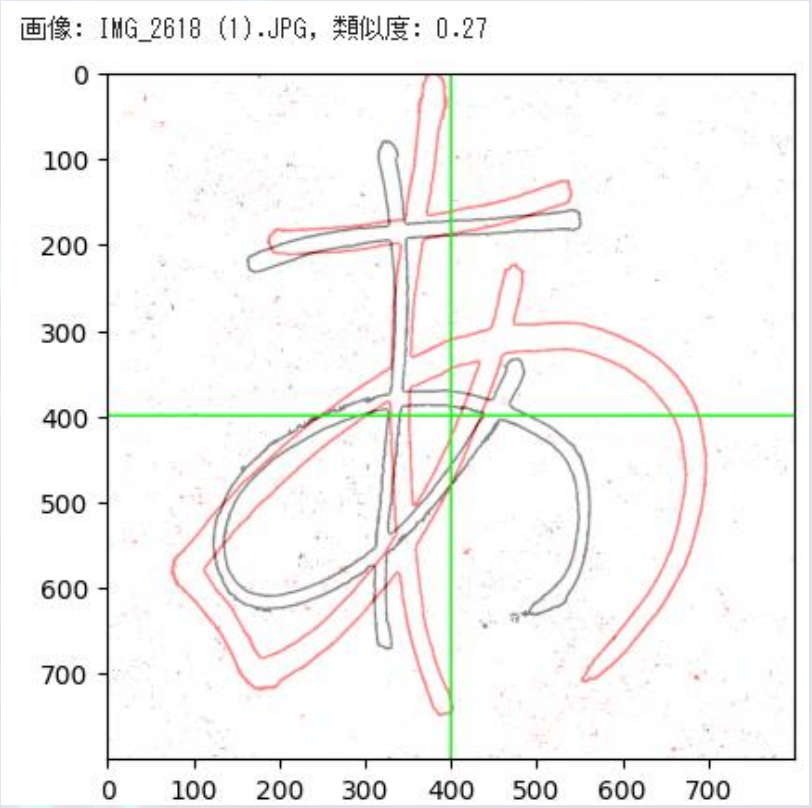
類似度　0.65



類似度　0.38

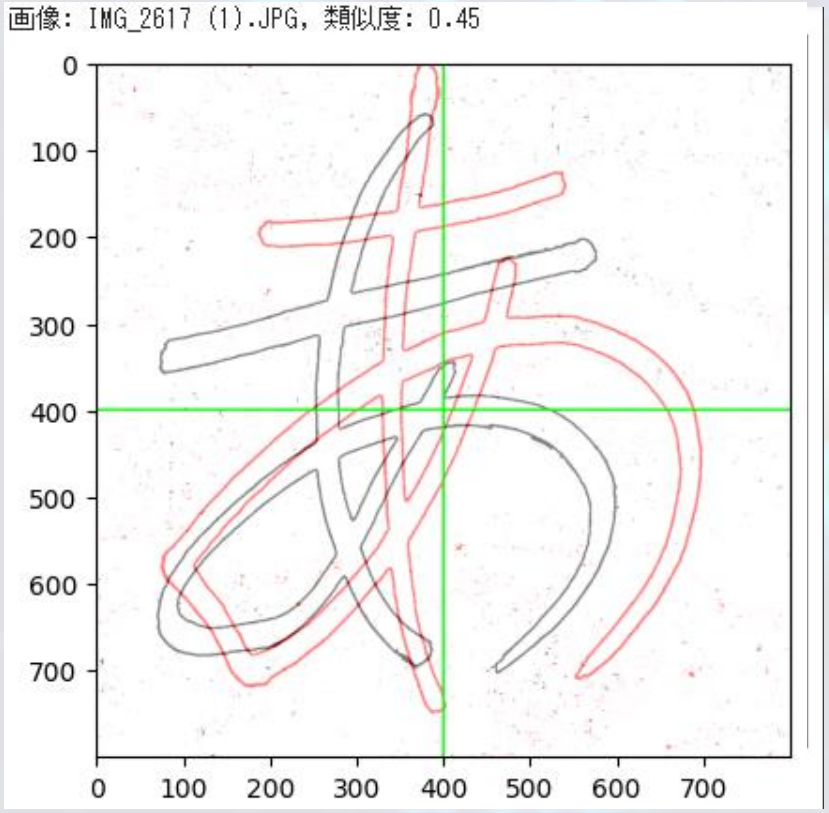


類似度　0.27

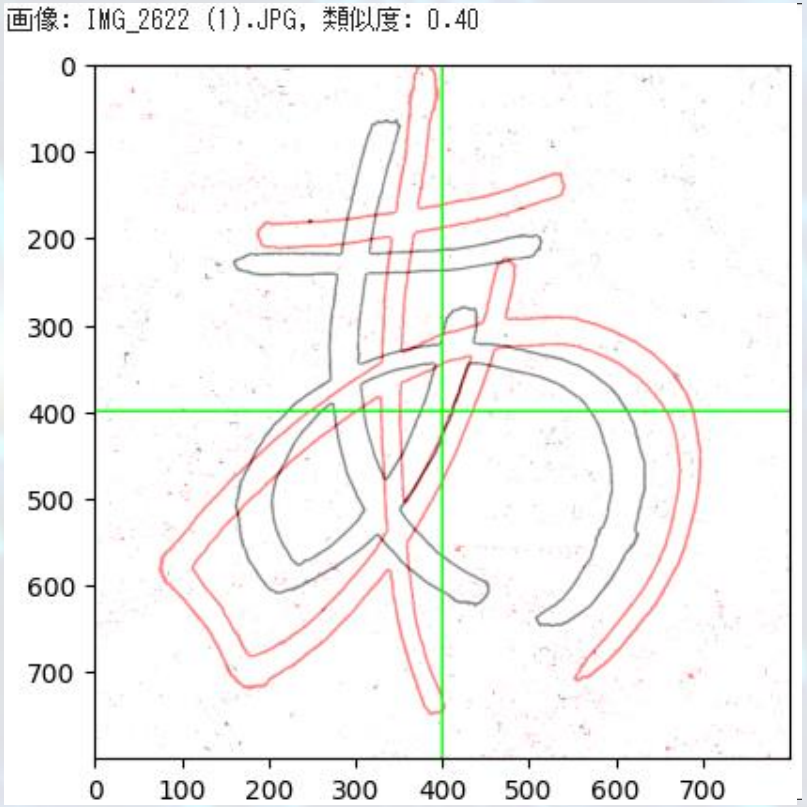


08 実験結果

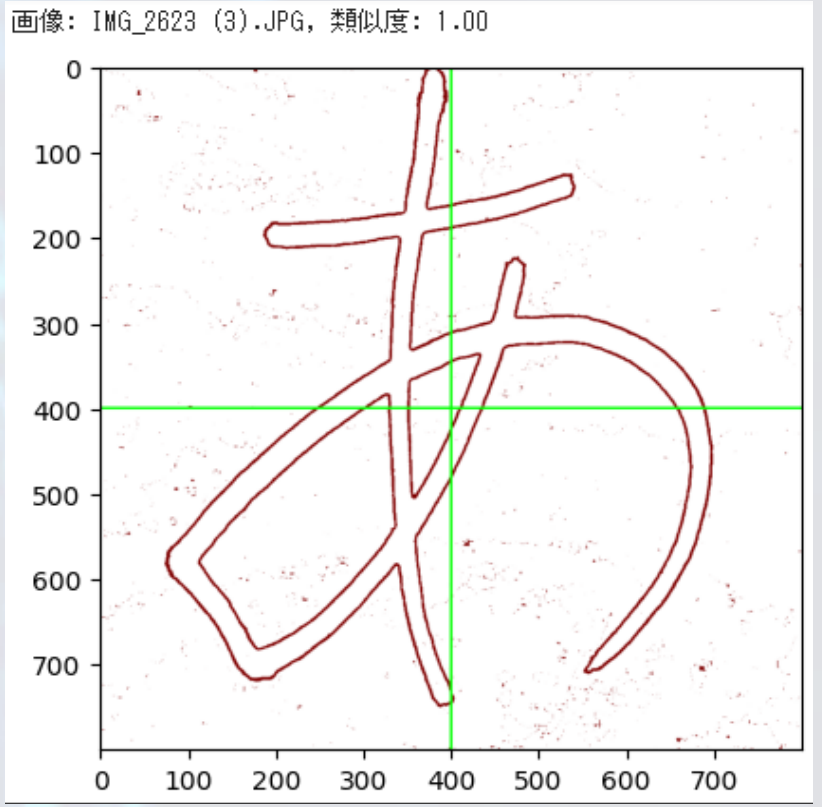
類似度　0.45



類似度　0.40



類似度　1.00



09 考察

- 主観ではきれいだと思っていた字を、実際にプログラムを実行してみると低い値が出た

- 自分たちが考えていたきれいな文字の定義とプログラム上での定義が違っているのではないか

- 文字自体が綺麗でも文字を書く時の位置や線の太さが関わってくるのではないか

10 今後の展望

- データ数を増やして、傾向を探る
- 文字のバリエーションを増やす（例：丸文字など）
- コード実行が成功したら、漢字やカタカナで試す

11　参考文献・謝辞

- Deep Learningを用いた印象評価推定AIの作成と検証(論文)
- https://qiita.com/FukuharaYohei/items/db88a8f4c4310afb5a0d　(2023/3/23)
- https://chigusa-web.com/blog/opencvsharp%E3%81%A7%E7%94%BB%E5%83%8F%E5%87%A6%E7%90%86/　(2023/6/19)
- https://marunouchi-tech.i-studio.co.jp/4930/　(2023/7/11)

惣田彩可さん、石崎秀晃さん、森本克己さん、奥田尚さん
ご指導ありがとうございました。