きれいな文字を書くためには

徳島県立脇町高等学校
高橋美優　町田若菜　住吉由莉佳

研究動機

　文字をきれいに書きたいと思い、どのように書いたら文字がきれいに見えるのかということに興味をもっていた。しかし、基本的に文字のきれいさには主観がはいってしまう。そこで、文字のきれいさを表したり、修正のためのフィードバックができるシステムを作りたいと考えた。私たちはGoogle Colabolatoryの環境でpythonを使ったプログラムでこれを実現したいと考え、生成AIであるChatGPTを活用しながら開発を行った。

研究指針

　お手本となる文字と比較したい文字について、それぞれの画像を四分割し、部分ごとの類似度を出力する。さらにお手本の文字を重ねて赤く表示させる。
①画像の読み込み
②画像の前処理関数
③画像の分割と文字部分の割合計算
④画像比較と評価関数
⑤結果の可視化関数

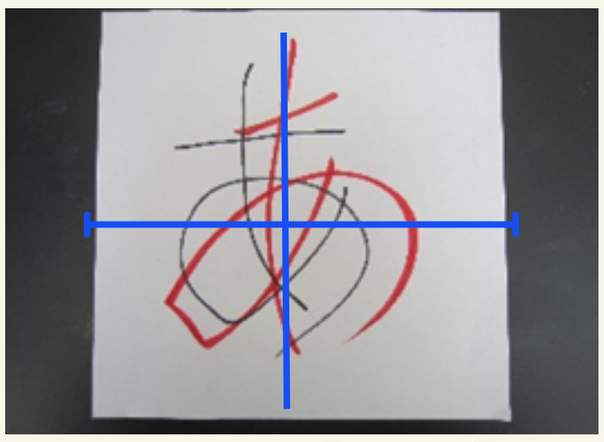

図1　イメージ図

Chat GPTのプロンプト

①画像の前処理関数
　お手本となる画像と比較する画像に対して共通の前処理を行う関数を定義する。
②画像の分割と文字部分の関数
　画像を四分割し、それぞれの領域において文字(黒)部分が占める面積の割合を計算する。この処理を行う関数を定義する。
③画像比較と評価関数
　お手本となる画像と比較する画像の各領域で文字が占める割合を比較し、指定された基準に基づいて評価する。
④結果の可視化関数
　最終的な結果を視覚的に確認するために使用

結果

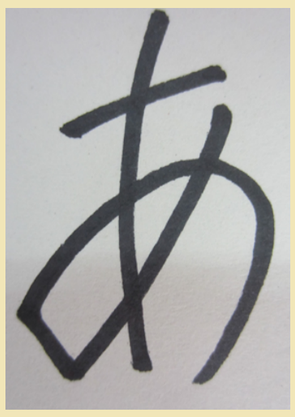

図2　お手本の文字

図3　比較した文字（きれい）

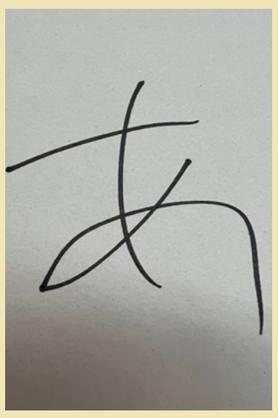

図4　比較した文字（へた）

①きれいな文字との比較
全体の類似度: 0.96
セクション 1 の類似度: 0.96
セクション 2 の類似度: 0.96
セクション 3 の類似度: 0.95
セクション 4 の類似度: 0.97

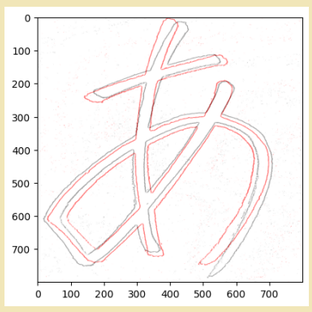

図5　出力図（きれい）

②へたな文字との比較
全体の類似度: 0.74
セクション 1 の類似度: 0.81
セクション 2 の類似度: 0.77
セクション 3 の類似度: 0.64
セクション 4 の類似度: 0.71

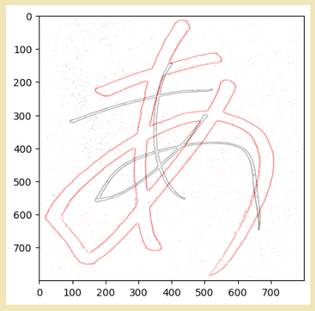

図6　出力図（へた）

考察

　お手本ときれいな文字、お手本とへたな文字の比較した値についてそれぞれの値の変化が乏しいため、結果が正しいとは言えないのではないか。
　考えられる理由として
①コードの関数が正しくない（計算方法等）
②文字の大きさがきれいさに関係してしまっている
③写真に影などが写り込み、正確な判断をすることができなくなっている
　また、出力の方法についてもフィードバックするには見えずらいところがある。

今後について

・サンプルデータ量を増やし、コードの修正する。
　（実行後の結果が適切であるかを確認、修正する）
・数値と表示された画像をもとに人間にフィードバックする。
・フィードバックの結果を検証する。

作成したコード

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

def upload_image():
  uploaded = files.upload()
  name, data = next(iter(uploaded.items()))
  image = cv2.imdecode(np.frombuffer(data, np.uint8), cv2.IMREAD_COLOR)
  return image, name

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

def calculate_image_similarity(image1, image2):
  # ピクセル値を正規化
  image1_normalized = image1.astype(float) / 255
  image2_normalized = image2.astype(float) / 255
  # 絶対値差分を計算
  difference = np.abs(image1_normalized - image2_normalized)
  # 類似度を計算（差分が少ないほど類似度が高い）
  similarity = 1 - np.mean(difference)
  return similarity

def split_image_into_sections(image):
  height, width = image.shape[:2]
  mid_height, mid_width = height // 2, width // 2
  # 画像を四分割
  top_left = image[:mid_height, :mid_width]
  top_right = image[:mid_height, mid_width:]
  bottom_left = image[mid_height:, :mid_width]
  bottom_right = image[mid_height:, mid_width:]
  return top_left, top_right, bottom_left, bottom_right

def calculate_section_similarity(image1, image2):
  sections1 = split_image_into_sections(image1)
  sections2 = split_image_into_sections(image2)
  similarities = []
  # 各セクションの類似度を計算
  for section1, section2 in zip(sections1, sections2):
    similarity = calculate_image_similarity(section1, section2)
    similarities.append(similarity)
  # 全セクションの類似度の平均を計算
  overall_similarity = sum(similarities) / len(similarities)
  return overall_similarity, similarities

def visualize_comparison(reference_image, target_image):
  alpha = 0.6
  # グレースケールからカラー画像に変換
  reference_image_color = cv2.cvtColor(reference_image, cv2.COLOR_GRAY2BGR)
  target_image_color = cv2.cvtColor(target_image, cv2.COLOR_GRAY2BGR)
  # お手本画像の赤色チャンネルを最大に設定
  reference_image_color[:, :, 2] = 255
  # 合成画像を作成
  output = cv2.addWeighted(reference_image_color, alpha, target_image_color, 1 - alpha, 0)
  plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
  plt.show()


# 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_image, ref_name = upload_image()
ref_image_processed = preprocess_image(ref_image)

print("比較する画像をアップロードしてください。")
target_image, target_name = upload_image()
target_image_processed = preprocess_image(target_image)

# 全体の類似度とセクションごとの類似度を計算
overall_similarity, section_similarities = calculate_section_similarity(ref_image_processed, target_image_processed)

# 結果の表示
print(f"全体の類似度: {overall_similarity:.2f}")
for i, sim in enumerate(section_similarities, 1):
  print(f"セクション {i} の類似度: {sim:.2f}")

# お手本画像と比較画像を赤くして重ねて表示
visualize_comparison(ref_image_processed, target_image_processed)
```

関連文献
•Deep Learningを用いた印象評価推定AIの作成と検証(山田悟史、2019）
•特徴量選択の基本まとめ考文献には多くのスペースが必要となるため、研究に使用した主要文献のみを引用します。　(2023/3/23)
・OpenCVSharpで画像処理！
　考文献には多くのスペースが必要となるため、研究に使用した主要文献のみを引用します。　(2023/6/19)
•GoogleColaboratoryで捗る画像処理～OpenCVの導入～
　考文献には多くのスペースが必要となるため、研究に使用した主要文献のみを引用します。　(2023/7/11)