

2025年4月29日

卒研究生課題 3a

画像検索 CGI 課題 画像特徴量の抽出と画像検索 CGI の作成

2210593 松浦史明

1. Introduction

課題 3a の特微量ファイルを読み込み画像間の距離を計算する画像検索 CGI システムとして、画像 400 枚程度をあらかじめ用意した上で、<https://mm.cs.uec.ac.jp/matsuura-f/imsearch/> を作成した。

2. Methods

2.1 CGI システムの概要

CGI システムは、段階的詳細化の手法を用いて、フロントエンドから順に関数設計を行った。フロントエンドは html と JavaScript、CSS を用いることとし、バックエンドでは CGI を用いた。フロントエンドとバックエンド間の通信は、データ量削減のために json 形式による画像表示順のみをやりとりするものとし、類似度計算はバックエンドで、画像表示はフロントエンドでそれぞれ分担を行った。

2.1.1 フロントエンド

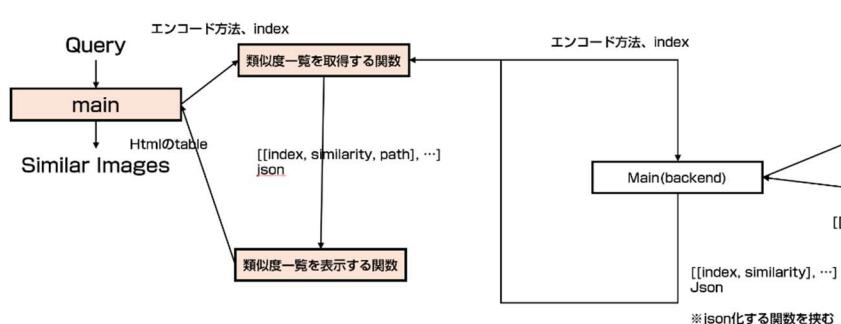


図 1 フロントエンドの概略図

動作全体を、エンコード方式と距離計算方式を受け取り、画像を表示する関数と定義し、段階的詳細化により 2 つの関数へ分離した。1 つはエンコード方式と距離計算方式、類似度を計算したい画像の index を http の GET メソッドのパラメータを用いて CGI のメイン関数へ渡す関数、もう 1 つは GET のレスポンスとして Json 形式の画像表示順を含んだデータを受け取り、それを表示する関数である。実際の実装では前者を `fetchAndRender();` が、その他を `document.addEventListener` 内の `fetchAndRender();` 以外の関数群が担っている。

2.1.2 バックエンド

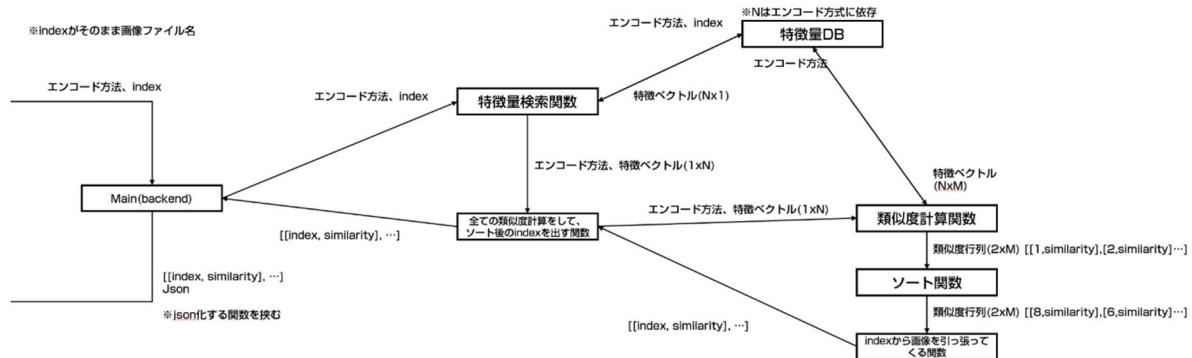


図 2 バックエンドの概略図

動作全体をエンコード方式と距離計算方式、類似度を計算したい画像の index を受け取り、画像を表示する順序を含む配列を返す関数と定義し、段階的詳細化により複数の関数へ分離した。

まず、画像は全て一度種々のエンコード方式でエンコードを行い、その行列データを特微量データベースとして保存を行った。次に、その行列データから Query となるものを取り出し、元のデータベースとの間で距離計算を行う。最後に、距離計算データの並び替えを行い、どの画像をどの順番で表示すれば良いかを json 形式で出力を行った。

2.2 画像のエンコード

画像のエンコードは/py/encode.py を用いて一括で行った。

2.2.1 用意した画像

合計で 406 枚の画像を用意した。画像はクローラーでインターネット上から「うどん」「そば」「京都」「浅草」の単語に関連する画像から収集を行い、1.jpg, 2.jpg, ... のように連番画像とした。今回 CGI 上での画像やエンコード情報は全てリストに一度格納する必要があり、その index は 0 から始まることから、index に 1 を加えたものの画像ファイルを表示するよう設計を行った。

2.2.2 ヒストグラムによるエンコード

ヒストグラムによるエンコードでは、入力画像を RGB、HSV または LUV カラー空間へ変換した上で要素ごとに分割を行い、指定された bin の値と一致する画素値を数える作業を行った。

Encode.hist() や Encode.hist_partition() では numpy のみを用いた完全実装を、Encode.hist_partition_new() では opencv を併用した実装を行った。Numpy のみの実装では画素値ごとに for ループを構成し、numpy のマスク生成機能を用いて、画素値がループの i と一致する場合は True(1) を、そうでない場合は False(0) となるマップを生成し、その行列の和を一度に計算することで、各画素値と一致する画素数をカウントすることにより、エンコードを行うこととした。

OpenCV を用いた関数では、cv2.calcHist() を用いてヒストグラムを作成した。

Numpy のみでの完全実装では動作はしたもの、エンコード速度が現実的でなかったことから、実際のエンコードでは opencv を用いた関数を使用した。

2.2.3 ヒストグラムへ分割を追加したエンコード

次に、前項で作成したヒストグラムエンコードを、画像全体を 2×2 または 3×3 で分割したそれぞれの箇所で行うことにより、元々のヒストグラム特徴ベクトルの 4 倍または 9 倍の特徴ベクトルを作成した。基本的な実装は前項で作成したものと変化なく、`Encode.hist_partition_new()`へ分割サイズを入力することで、計算するものとした。

2.2.4 DCNN 特徴量によるエンコード

最後に、DCNN 特徴量によりエンコードを行った。ネットワークは VGG16 を用いて、最終そうである `fc7` の出力をそのまま特徴ベクトルとした。実装では `do_encode_dcnn()` により、エンコードを行った。

2.3 CGI システムでの距離計算

距離計算はすべて、query ベクトルと特徴量データベースを受け取って、距離と index が格納された 2 次元配列を返す関数として、`index.cgi` の上で実装を行った。

距離計算はヒストグラムインターフェクション (`calc_channel_intersection` 関数) とユークリッド距離 (`calc_euclidean_distance` 関数) の 2 種類での計算を、フロントエンドの選択によってどちらか片方を行うものとした。

2.4 バックエンドからフロントエンドへの Json データ

バックエンドからフロントエンドは、計算されソートされた配列より、index と距離の値、また index に 1 を加えることで求めることのできる、画像の実体があるパスの 3 つを 1 つのグループとして、表示順に画像分だけ並べたものを json として整形し、GET のレスポンスとしてフロントエンドへ渡す作業を行った。

2.5 フロントエンドでの画像の表示

前項で作成し、レスポンスとして返送された Json データを用いて、画像クリック時に再計算が行われるように `cgi` のパラメータ埋め込みを行ったうえで、table へ画像を挿入し、表示させた。

3. Result

結果、意図した通りに画像を表示させることができた。ここでは次のクエリ画像



図3 クエリ画像(index 148)

を例に、top10の画像を用いて、結果の推移を確認していく。

3.1.1 RGB (各色 64 ビン)



図4 RGB(64 ビン)の場合の top10

3.1.2 RGB (各色 64 ビン、3x3 分割)



図5 RGB(64 ビン、3x3 分割)の場合の top10

3.1.3 HSV (各色 64 ビン)



図6 HSV(64 ビン)の場合の top10

3.1.4 HSV (各色 64 ビン、3x3 分割)



図 7 HSV(64 ビン、3x3 分割)の場合の top10

3.1.5LUV (各色 64 ビン)



図 8 LUV(64 ビン)の場合の top10

3.1.6LUV (各色 64 ビン、3x3 分割)



図 9 LUV(64 ビン、3x3 分割)の場合の top10

3.1.7VGG16 fc7



図 10 VGG16 の場合の top10

4. Discussion

4.1 ヒストグラム間の手法の比較

まず、ヒストグラムで分割を行わない場合と行う場合では、行った場合が特徴ベクトルが増え、物体の位置による表現能力が向上したことから、より近しい物体が列挙できていることが確認できる。逆に分割を行わない場合は、単純に画素値の傾向によって距離が決定されるため、全く異なる物体が写っていても、類似物体として表示される結果となった。

次に、RGB、HSV、LUV のそれぞれを比較した場合、LUV がもっともらしい結果となることが確認できた。これは、LUV 色空間が XYZ から色差の均等性を目指して開発されたものであることが影響しているのではないかと考えることができる

4.2 ヒストグラムと DCNN 特徴との間の比較

ヒストグラムと DCNN を比較した場合では、DCNN 特徴がよく特徴を掴むことができ、もっともらしい結果を出力することができる事が分かった。これは、vgg16 が元々 1000 種類物体認識に

よって pre-training されていることから、物体の特徴をよく捉えるような重みが設定されていることから、特徴を効率よく抽出可能となっていることからであると推測できる。

5. Appendix

5.1 ディレクトリ構造

以下のディレクトリ構造と役割で、ページを作成した。

```
Imsearch/
  features/ …エンコードした行列を、.npy ファイルとして格納している
  img/     …分類に用いる画像を格納している
  py/      …エンコードに用いた python ファイルを格納している
    encode.py
  static/   …静的ファイルである javascript と css を格納している
    css/
      main.css
    js/
      main.js
  index.html…ページ本体のファイル
  main.cgi  …python で記述された、cgi 本体のファイル
```

5.2 encode.py

```
from pathlib import Path

import cv2
import numpy as np

from PIL import Image
import torch
import torchvision.models as models
import torch.nn as nn
import glob

class Encode:
    # path を受け取って、3Nx1 の特徴行列を出す関数
    @staticmethod
    def hist(type: str, path2img: str, num_bin: int) -> np.ndarray:
        # 画像読み込み
        img = cv2.imread(path2img, cv2.IMREAD_COLOR)
        if img is None:
            raise FileNotFoundError(f"can't read {path2img}")

        match type:
            case "hist_rgb":
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            case "hist_hsv":
                img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            case "hist_luv":
                img = cv2.cvtColor(img, cv2.COLOR_BGR2LUV)

        # rgb へ分解
        r = img[:, :, 0]
```

```

g = img[:, :, 1]
b = img[:, :, 2]

# hist 用
hist_r = np.zeros((num_bin,), dtype=np.int64)
hist_g = np.zeros((num_bin,), dtype=np.int64)
hist_b = np.zeros((num_bin,), dtype=np.int64)

bin_interval = 256 // num_bin

for i in range(0, 256):
    # 画素値が i と一致するところは True(1)、それ以外は 0
    map_r = (r == i)
    map_g = (g == i)
    map_b = (b == i)
    # np.sum で 1 の部分を一気にカウント
    sum_r = np.sum(map_r)
    sum_g = np.sum(map_g)
    sum_b = np.sum(map_b)
    # hist 配列に代入
    hist_r[i // bin_interval] += sum_r
    hist_g[i // bin_interval] += sum_g
    hist_b[i // bin_interval] += sum_b

# hist を合成
output = np.concatenate([hist_r, hist_g, hist_b], axis=0)

return output

@staticmethod
def hist_partition(type: str, path2img: str, num_bin: int, grid: tuple[int,int] = (2,2)) -> np.ndarray:
    # 画像読み込み
    img = cv2.imread(path2img, cv2.IMREAD_COLOR)
    if img is None:
        raise FileNotFoundError(f"can't read {path2img}")

    match type:
        case t if t.startswith("hist_rgb"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        case t if t.startswith("hist_hsv"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        case t if t.startswith("hist_luv"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2LUV)

    # bit 数取得
    img_num_bit = np.iinfo(img.dtype).max

    H, W, _ = img.shape
    gy, gx = grid
    h_step = H // gy
    w_step = W // gx

    # ここでリストにためて、最後に一度結合する
    feats = []

    # 2) サブ領域ごとにヒスト計算
    for iy in range(gy):
        for ix in range(gx):
            sub_img = img[iy*h_step:(iy+1)*h_step, ix*w_step:(ix+1)*w_step]

            # rgb へ分解
            r = sub_img[:, :, 0]
            g = sub_img[:, :, 1]
            b = sub_img[:, :, 2]

            # hist 用
            hist_r = np.zeros((num_bin,), dtype=np.int64)
            hist_g = np.zeros((num_bin,), dtype=np.int64)
            hist_b = np.zeros((num_bin,), dtype=np.int64)

            # bin_interval = 256 // num_bin
            bin_interval = (img_num_bit + 1) // num_bin

            for i in range(0, img_num_bit + 1):
                # 画素値が i と一致するところは True(1)、それ以外は 0
                map_r = (r == i)
                map_g = (g == i)

```

```

map_b = (b == i)
# np.sum で 1 の部分を一気にカウント
sum_r = np.sum(map_r)
sum_g = np.sum(map_g)
sum_b = np.sum(map_b)
# hist 配列に代入
hist_r[i // bin_interval] += sum_r
hist_g[i // bin_interval] += sum_g
hist_b[i // bin_interval] += sum_b

# hist を合成
feats.append(np.concatenate([hist_r, hist_g, hist_b], axis=0))

feature_martix = np.concatenate(feats, axis=0).astype(np.float32)
s = feature_martix.sum()
if s > 0:
    feature_martix /= s
return feature_martix

@staticmethod
def hist_partition_new(type: str, path2img: str, num_bin: int, grid: tuple[int,int]) -> np.ndarray:
    img = cv2.imread(path2img, cv2.IMREAD_COLOR)
    if img is None:
        raise FileNotFoundError(f"can't read {path2img}")

    # カラースペース変換
    match type:
        case t if t.startswith("hist_rgb"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        case t if t.startswith("hist_hsv"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        case t if t.startswith("hist_luv"):
            img = cv2.cvtColor(img, cv2.COLOR_BGR2LUV)

    H, W, _ = img.shape
    gy, gx = grid
    h_step = H // gy
    w_step = W // gx

    feats: list[np.ndarray] = []
    for iy in range(gy):
        for ix in range(gx):
            # sub = img[iy*h_step:(iy+1)*h_step, ix*w_step:(ix+1)*w_step]
            y0 = iy*h_step; y1 = (H if iy==gy-1 else (iy+1)*h_step)
            x0 = ix*w_step; x1 = (W if ix==gx-1 else (ix+1)*w_step)
            sub = img[y0:y1, x0:x1]

            # 各チャンネル毎にヒストグラム抽出
            # H は [0,180]、S,V,L,U は [0,256] でヒストを切る
            if type.startswith("hist_hsv"):
                h_r = cv2.calcHist([sub], [0], None, [num_bin], [0, 180]).flatten() # Hue
                h_g = cv2.calcHist([sub], [1], None, [num_bin], [0, 256]).flatten()
                h_b = cv2.calcHist([sub], [2], None, [num_bin], [0, 256]).flatten()
            else:
                # RGB / LUV はすべて [0,256]
                h_r = cv2.calcHist([sub], [0], None, [num_bin], [0, 256]).flatten()
                h_g = cv2.calcHist([sub], [1], None, [num_bin], [0, 256]).flatten()
                h_b = cv2.calcHist([sub], [2], None, [num_bin], [0, 256]).flatten()

            # チャンネルごと L1 正規化
            # h_r /= (h_r.sum() + 1e-9)
            # h_g /= (h_g.sum() + 1e-9)
            # h_b /= (h_b.sum() + 1e-9)

            vec = np.concatenate([h_r, h_g, h_b]).astype(np.float32)

            # 全体で L1 正規化
            vec /= (vec.sum() + 1e-9)

            feats.append(vec)

    return np.concatenate(feats, axis=0)

def do_encode_hist_partition(method, num_bin, grid):
    # 特微量行列 (最終的に、3NxM)

```

```

features_list = []

i = 0
while True:
    path2img = Path(f"../img/{i+1}.jpg")
    if not path2img.exists():
        print(f"{path2img} doesn't exist")
        break

    # 各種 Encode
    # vec = Encode.hist(METHOD, str(path2img), num_bin) # 3Nx1
    # vec = Encode.hist_partition(method, str(path2img), num_bin, grid)
    vec = Encode.hist_partition_new(method, str(path2img), num_bin, grid)

    # feature_matrix = np.concatenate([feature_matrix, vec.reshape(-1, 1)], axis=1)
    features_list.append(vec)

    print(f"Encode {i+1}.jpg finished")
    i = i + 1

feature_matrix = np.stack(features_list, axis=1)

np.save(f"../features/features_{method}.npy", feature_matrix)
print(f"Saved features_{method}.npy ({feature_matrix.shape})")

def do_encode_dcnn(method):
    device = 'cuda' if torch.cuda.is_available() else 'cpu'

    weights=models.VGG16_Weights.DEFAULT
    vgg16 = models.vgg16(weights=weights,progress=True)
    vgg16fc7 = torch.nn.Sequential(
        vgg16.features,
        vgg16.avgpool,
        nn.Flatten(),
        *list(vgg16.classifier.children())[-3] # 最後の 3 つの layer(relu,dropout,fc1000)を削除
    )
    vgg16fc7 = vgg16fc7.to(device)
    vgg16fc7.eval()

    # imglist = sorted(glob.glob("../img/*jpg"))
    imglist = []
    idx = 0
    while True:
        fname = f"../img/{idx+1}.jpg"
        if not Path(fname).exists():
            break
        imglist.append(fname)
        idx += 1

    in_size=224
    imgs = np.empty((0,in_size,in_size,3), dtype=np.float32)
    for i,img_path in enumerate(imglist):
        if i%100==0:
            print("reading {}th image".format(i))
        x = np.array(Image.open(img_path).resize((in_size,in_size)), dtype=np.float32)
        x = np.expand_dims(x, axis=0)
        imgs = np.vstack((imgs,x))
    mean=np.array([0.485, 0.456, 0.406], dtype=np.float32)
    std=np.array([0.229, 0.224, 0.225], dtype=np.float32)
    imgs=(imgs/255.0-mean)/std
    imgs=imgs.transpose(0,3,1,2) # HWC -> CHW
    img=torch.from_numpy(imgs)
    print(imgs.shape)

    with torch.no_grad():
        fc=vgg16fc7(img.to(device)).cpu().numpy()
        print(fc.shape) # shape の表示

    feature_matrix = fc.T

    np.save(f"../features/features_{method}.npy", feature_matrix)
    print(f"Saved features_{method}.npy ({feature_matrix.shape})")

if __name__ == "__main__":
    do_encode_hist_partition("hist_rgb_256", 256, (1, 1))

```

```

do_encode_hist_partition("hist_rgb_64", 64, (1, 1))
do_encode_hist_partition("hist_rgb_64_2x2", 64, (2, 2))
do_encode_hist_partition("hist_rgb_64_3x3", 64, (3, 3))

do_encode_hist_partition("hist_hsv_256", 256, (1, 1))
do_encode_hist_partition("hist_hsv_64", 64, (1, 1))
do_encode_hist_partition("hist_hsv_64_2x2", 64, (2, 2))
do_encode_hist_partition("hist_hsv_64_3x3", 64, (3, 3))

do_encode_hist_partition("hist_luv_256", 256, (1, 1))
do_encode_hist_partition("hist_luv_64", 64, (1, 1))
do_encode_hist_partition("hist_luv_64_2x2", 64, (2, 2))
do_encode_hist_partition("hist_luv_64_3x3", 64, (3, 3))

do_encode_dcnn("vgg16_fc7")

```

5.3 main.cgi

```

#!/usr/local/anaconda3/bin/python3
# -*- coding: utf-8 -*-

import cgi
import cgitb
import json
import os
from pathlib import Path

import numpy as np

# デバッグ ON
cgitb.enable()

# メソッドごとの .npy ファイル名だけを保持
METHODS = {
    "hist_rgb_256": "features_hist_rgb_256.npy",
    "hist_rgb_64": "features_hist_rgb_64.npy",
    "hist_rgb_64_2x2": "features_hist_rgb_64_2x2.npy",
    "hist_rgb_64_3x3": "features_hist_rgb_64_3x3.npy",
    "hist_hsv_256": "features_hist_hsv_256.npy",
    "hist_hsv_64": "features_hist_hsv_64.npy",
    "hist_hsv_64_2x2": "features_hist_hsv_64_2x2.npy",
    "hist_hsv_64_3x3": "features_hist_hsv_64_3x3.npy",
    "hist_luv_256": "features_hist_luv_256.npy",
    "hist_luv_64": "features_hist_luv_64.npy",
    "hist_luv_64_2x2": "features_hist_luv_64_2x2.npy",
    "hist_luv_64_3x3": "features_hist_luv_64_3x3.npy",
    "vgg16_fc7": "features_vgg16_fc7.npy",
}

# 保存済み.npy を読み込んで、特徴量行列を返す関数
def load_feature_db(method: str) -> np.ndarray:
    if method not in METHODS:
        raise ValueError(f"Unsupported method: {method}")

    base = Path(__file__).resolve().parent.parent / "imsearch"
    feat_path = base / "features" / METHODS[method]
    if not feat_path.exists():
        raise FileNotFoundError(f"{feat_path} がありません")

    features = np.load(feat_path, mmap_mode="r")
    return features

# 以下類似度計算関数
# L2
def calc_euclidean_distance(query_vec: np.ndarray, db: np.ndarray) -> np.ndarray:
    diff = db - query_vec[:, None]
    sims = np.linalg.norm(diff, axis=0)
    return sims

# L1
def calc_hist_intersection(query_vec: np.ndarray, db: np.ndarray) -> np.ndarray:
    return np.minimum(db, query_vec[:, None]).sum(axis=0)
    # numer = np.minimum(db, query_vec[:, None]).sum(axis=0)

```

```

# denom = query_vec.sum() + 1e-9
# return numer / denom

# チャンネル別平均 intersection
def calc_channel_intersection(query_vec: np.ndarray, db: np.ndarray, bins: int) -> np.ndarray:
    sims = []
    for c in range(3):
        q_c = query_vec[c*bins:(c+1)*bins]
        d_c = db[c*bins:(c+1)*bins]
        numer = np.minimum(q_c[:, None], d_c).sum(axis=0)
        denom = q_c.sum() + 1e-9
        sims.append(numer/denom)
    return np.mean(sims, axis=0)

# 以下ソート関数
def sort_indices_by_desc(sims: np.ndarray) -> np.ndarray:
    return np.argsort(sims)[::-1]

def sort_indices_by_distance(sims: np.ndarray) -> np.ndarray:
    return np.argsort(sims) # 小さいものが先

# 結果を json 化する関数
def build_results(sorted_idx: np.ndarray, sims: np.ndarray) -> list:
    results = []
    for i in sorted_idx:
        results.append({
            "index": int(i),
            "similarity": float(sims[i]),
            "path": f"./img/{i+1}.jpg" # パスに注意 (current はどうやら/www/imsearch 抜いらしゃい)
        })
    return results

# encode 方法、dist 方法、index を受け取って、結果が格納された json を返す関数
def main(method, index, dist_method):
    try:
        # DB やらの読み込み
        db = load_feature_db(method)
        D, M = db.shape
        if not (0 <= index < M):
            raise IndexError(f"index は 0～{M-1} の間で指定してください")
        query_vec = db[:, index]

        # 距離計算
        if method.startswith("hist_"):
            if dist_method == "intersect":
                bins = db.shape[0] // 3
                sims = calc_channel_intersection(query_vec, db, bins)
                sorted_idx = sort_indices_by_desc(sims)
            elif dist_method == "euclid":
                sims = calc_euclidean_distance(query_vec, db)
                sorted_idx = sort_indices_by_distance(sims)
            else:
                raise ValueError(f"Unsupported dist_method: {dist_method}")
        else:
            sims = calc_euclidean_distance(query_vec, db)
            sorted_idx = sort_indices_by_distance(sims)

        # レスポンスの組み立て
        result = build_results(sorted_idx, sims)
        print(json.dumps(result, ensure_ascii=False))

    except Exception as e:
        # エラー時も JSON で返す
        print(json.dumps({"error": str(e)}, ensure_ascii=False))

if __name__ == "__main__":
    # CGI ヘッダ
    print("Content-Type: application/json; charset=utf-8")
    print()
    form = cgi.FieldStorage()
    method = form.getFirst("method", "hist_rgb_256")
    try:

```

```

index = int(form.getFirst("index", "0"))
except ValueError:
    index = 0

dist_method = form.getFirst("dist_method", "intersect")

# デバッグ用
# method = "hist_rgb"
# index = 1

main(method, index, dist_method)

```

5.4 main.js

```

document.addEventListener('DOMContentLoaded', () => {
    const params = new URLSearchParams(window.location.search);
    let method = params.get('method') || 'hist_rgb_256';
    let index = parseInt(params.get('index')) || '0', 10;
    let distMethod = params.get('dist_method') || 'intersect';

    const table = document.getElementById('results');
    const links = document.querySelectorAll('.method-link');

    // メソッドのリンク
    links.forEach(a => {
        const m = a.dataset.method;
        a.classList.toggle('active', m === method);
        a.addEventListener('click', e => {
            e.preventDefault();
            method = m;
            history.replaceState(null, '', `?method=${method}&index=${index}`);
            fetchAndRender();
            links.forEach(x => x.classList.toggle('active', x.dataset.method === method));
        });
    });

    // 画像クリックで再検索
    table.addEventListener('click', e => {
        if (e.target.tagName === 'IMG') {
            e.preventDefault();
            const fileNo = parseInt(e.target.alt.replace('Image ', ''), 10);
            index = fileNo - 1;
            history.replaceState(null, '', `?method=${method}&index=${index}`);
            fetchAndRender();
        }
    });

    // dist 計算方法のボタン用
    const intersectBtn = document.getElementById('intersect-btn');
    const euclidBtn = document.getElementById('euclid-btn');
    // active 付与
    if (distMethod === 'intersect') {
        intersectBtn.classList.add('active');
    } else {
        euclidBtn.classList.add('active');
    }
    // EventListener
    intersectBtn.addEventListener('click', e => {
        e.preventDefault();
        distMethod = 'intersect';
        intersectBtn.classList.add('active');
        euclidBtn.classList.remove('active');
        history.replaceState(null, '', `?method=${method}&index=${index}&dist_method=${distMethod}`);
        fetchAndRender();
    });
    euclidBtn.addEventListener('click', e => {
        e.preventDefault();
        distMethod = 'euclid';
        euclidBtn.classList.add('active');
        intersectBtn.classList.remove('active');
        history.replaceState(null, '', `?method=${method}&index=${index}&dist_method=${distMethod}`);
        fetchAndRender();
    });
}

```

```

// 初回描画
fetchAndRender();

function fetchAndRender() {
  table.innerHTML = '';
  const endpoint = `/matsuura-f/imsearch/main.cgi?method=${method}&index=${index}&dist_method=${distMethod}`;
  fetch(endpoint)
    .then(res => res.json())
    .then(data => {
      if (!Array.isArray(data)) {
        const msg = data.error || '想定外のレスポンスです';
        table.insertAdjacentHTML('beforebegin',
          `<p style="color:red">${msg}</p>`);
        return;
      }
      let row, cellCount = 0;
      data.forEach(item => {
        if (cellCount % 10 === 0) row = table.insertRow();
        const cell = row.insertCell();

        const index = item.index;
        // index は 0 始まり、fileNo は 1 から開始
        const fileNo = index + 1;
        cell.innerHTML = `

<!-- パスに注意 (current はどうやら/www/imsearch 抜いらしい) --&gt;
&lt;a href="#"&gt;&lt;img src="./img/${fileNo}.jpg" alt="Image ${fileNo}"&gt;&lt;/a&gt;
&lt;div class="info"&gt;DB index: ${item.index}&lt;/div&gt;
&lt;!--&gt;
&lt;div class="info"&gt;File: ${fileNo}.jpg&lt;/div&gt;--&gt;
&lt;div class="info"&gt;Sim: ${item.similarity.toFixed(3)}&lt;/div&gt;
`;
        cellCount++;
      });
    })
    .catch(err =&gt; {
      console.error(err);
      table.insertAdjacentHTML('beforebegin',
        `&lt;p style="color:red"&gt;通信エラー : ${err}&lt;/p&gt;`);
    });
}
};

</pre>

```

5.5 index.html

```

<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8" />
  <title>Image Search</title>
  <link rel="stylesheet" href="./static/css/main.css" />
  <script defer src="./static/js/main.js"></script>
</head>
<body>
  <h1>Image Search</h1>

  <h2>Distance Methods</h2>
  <button id="intersect-btn">Intersection</button>
  <button id="euclid-btn">Euclid</button>
  <br>
  <br>

  <h2>Encode Methods</h2>
  <div class="section" id="encode-section">
    <!-- メソッド切り替えのリンク -->
    <nav id="method-selection">
      <a href="#" class="method-link" data-method="hist_rgb_256">RGB(256bin)</a>
      <br>
      <a href="#" class="method-link" data-method="hist_rgb_64">RGB(64bin)</a>
      <a href="#" class="method-link" data-method="hist_rgb_64_2x2">RGB(64bin_2x2)</a>
      <a href="#" class="method-link" data-method="hist_rgb_64_3x3">RGB(64bin_3x3)</a>
      <br>
      <a href="#" class="method-link" data-method="hist_hsv_256">HSV(256bin)</a>
      <br>
      <a href="#" class="method-link" data-method="hist_hsv_64">HSV(64bin)</a>
      <a href="#" class="method-link" data-method="hist_hsv_64_2x2">HSV(64bin_2x2)</a>
      <a href="#" class="method-link" data-method="hist_hsv_64_3x3">HSV(64bin_3x3)</a>
      <br>
    </nav>
  </div>

```

```
<a href="#" class="method-link" data-method="hist_luv_256">LUV(256bin)</a>
<br>
<a href="#" class="method-link" data-method="hist_luv_64">LUV(64bin)</a>
<a href="#" class="method-link" data-method="hist_luv_64_2x2">LUV(64bin_2x2)</a>
<a href="#" class="method-link" data-method="hist_luv_64_3x3">LUV(64bin_3x3)</a>
<br>
<a href="#" class="method-link" data-method="vgg16_fc7">VGG16_fc7</a>
</nav>
</div>

<br>
<br>

<!-- 結果の表 -->
<table id="results"></table>
</body>
</html>
```