Cel laboratorium

Celem laboratorium była poprawa zaimplementowanych 5 wybranych prostych deskryptorów kształtu przedstawionych przedstawianych prowadzącemu na poprzednich zajęciach oraz porównanie wyników..

Przebieg laboratorium

Po przedstawieniu prowadzącemu wyników z poprzedniego zadania, stwierdził on, że wykonane zadanie należy poprawić. Prowadzący wyjaśnił dokładnie jak zadanie powinno być wykonane, a celem laboratorium było poprawienie implementacji oraz porównanie wyników z poprzednich zajęć.

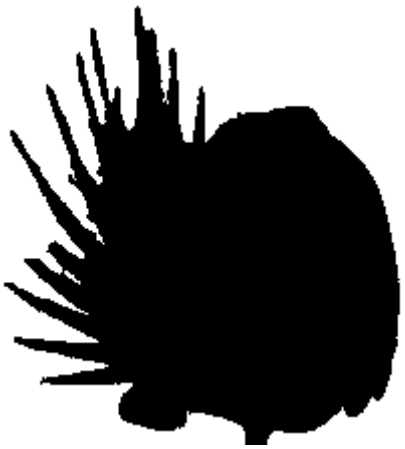
Dla każdej klasy wybrano po 1 obrazie jako obraz wzorcowy (część ucząca). Dla tych obrazów były obliczane wartości deskryptorów. Następnie dla pozostałych obrazów (część testowa) były również obliczane wartości deskryptorów.
Po dokonaniu obliczeń przez deskryptory dokonywany był pomiar odległości euklidesowej dla danego deskryptora dla każdego obrazu testowego względem wszystkich obrazów wzorcowych. Klasa obrazu wzorcowego z najniższą odległością odzwierciedlała tą predykcyjną.

Wybrane obrazy dla poszczególnych klas:

- Gambles quail

- Glossy ibis

- Greator sage grous

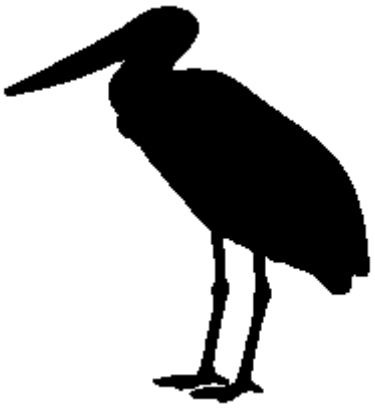


- Hooded merganser

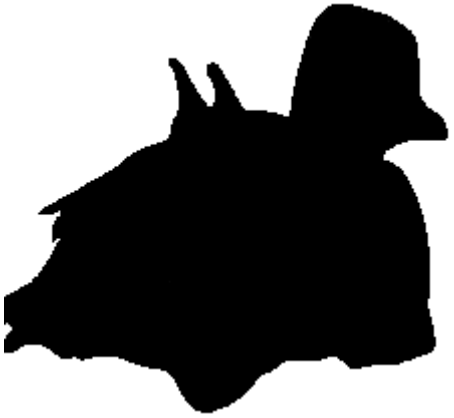
- Indian vulture

- Jabiru

- King eider



- Long eared owl



- Tit mouse

- Touchan

Kod programu

Importy oraz stałe:

```
from typing import Tuple
import numpy as np
import pandas as pd
import os
import cv2

IMAGES_DIR = './images'
```

Funkcje pomocnicze:

```
def read_images() -> dict:
    images = {'black': [], 'contour': []}

    dirs = [x[0] for x in os.walk(IMAGES_DIR)]
    black_and_white_dirs = get_list_of_dirs(dirs, 'black_and_white')

    for dir_index, black_and_white_dir in enumerate(black_and_white_dirs):
        filenames = [x[2] for x in os.walk(black_and_white_dir)][0]
        images['black'].append([])
        images['contour'].append([])

        for filename_iter, filename in enumerate(filenames):
            image = cv2.imread(f'{black_and_white_dir}/{filename}')
            images['black'][dir_index].append(convert_image_to_black(image))
            images['contour'][dir_index].append(convert_image_to_contour(image))

    return images


def select_representatives_numbers(images: dict) -> list:
    representative_images_number: dict = {
        'gambles-quail': 6,
        'glossy-ibis': 10,
        'greator-sage-grous': 5,
        'hooded-merganser': 1,
        'indian-vulture': 4,
        'jabiru': 1,
        'king-eider': 8,
        'long-eared-owl': 4,
        'tit-mouse': 8,
        'touchan': 6}  # from 1

    return list(map(lambda number: number - 1,
representative_images_number.values()))


def train_test_split(images: dict, representative_numbers: list) -> Tuple:
    train_images = {'black': [], 'contour': []}
    test_images = {'black': [], 'contour': []}

    for number_iter, number in enumerate(representative_numbers):
        train_images['black'].append(images['black'][number_iter][number])
        train_images['contour'].append(images['contour'][number_iter][number])

        test_images['black'].append([])
        test_images['contour'].append([])

        for image_index in range(len(images['black'])):
            if image_index == number:
                continue


test_images['black'][number_iter].append(images['black'][number_iter][image_index])

test_images['contour'][number_iter].append(images['contour'][number_iter][image_ind
ex])

    return train_images, test_images
```

```
def get_list_of_dirs(dirs: list, sub_dir_name: str = 'color'):
    list_dir = [x for x in dirs if sub_dir_name in x]
    list_dir = [x.replace("\\", "/") for x in list_dir]

    return list_dir


def convert_image_to_black(image: np.array):
    image_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    image_gray_inv = 255 - image_gray
    tresh, image_black_and_white = cv2.threshold(image_gray_inv, 0, 255,
cv2.THRESH_BINARY_INV)

    return image_black_and_white


def convert_image_to_contour(image: np.array):
    image_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    thresh, image_black_and_white = cv2.threshold(image_gray, 0, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    image_contour = image_black_and_white.copy()
    image_contour[:] = 255

    contours, hierarchy = cv2.findContours(image_black_and_white, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(image=image_contour, contours=contours, contourIdx=-1,
color=(0, 0, 0), thickness=1)

    return image_contour


def calc_descriptors_for_images(train_images: dict, test_images: dict) -> Tuple:
    train_results = {descriptor.__name__: [] for descriptor in descriptors}
    test_results = {descriptor.__name__: [] for descriptor in descriptors}

    for descriptor in descriptors:
        image_type = 'black' if descriptor.__name__ != perimeter_descriptor.__name__
else 'contour'

        descriptor_train_result = [descriptor(image) for image in
train_images[image_type]]
        train_results[descriptor.__name__] = descriptor_train_result

        descriptor_test_result = []
        for test_class_images in test_images[image_type]:
            test_result_class = [descriptor(test_image) for test_image in
test_class_images]
            descriptor_test_result.append(test_result_class)

        test_results[descriptor.__name__] = descriptor_test_result

    return train_results, test_results


def predict_based_on_distance(test_descriptor_result: float,
train_descriptor_results: list):
```

```
    results = np.array([calc_distance(train_descriptor_result,
test_descriptor_result) for train_descriptor_result in train_descriptor_results])

    return np.argmin(results)
```

Deskryptory:

```
def area_descriptor(image: np.array) -> float:
    return np.array(list(zip(*np.where(image == 0)))).shape[0]


def perimeter_descriptor(contour_image: np.array) -> float:
    return area_descriptor(contour_image)


def roundness_descriptor(image: np.array) -> float:
    return perimeter_descriptor(image) ** 2 / (4 * np.pi * area_descriptor(image))


def compactness_descriptor(image: np.array) -> float:
    return perimeter_descriptor(image) ** 2 / area_descriptor(image)


def eccentricity_descriptor(image: np.array) -> float:
    boundary = {'x_min': image.shape[0], 'x_max': 0, 'y_min': image.shape[1],
'y_max': 0}

    for x, row in enumerate(image):
        for y, val in enumerate(row):
            if val != 0:
                continue

            if x < boundary['x_min']:
                boundary['x_min'] = x

            if x > boundary['x_max']:
                boundary['x_max'] = x

            if y < boundary['y_min']:
                boundary['y_min'] = y

            if y > boundary['y_max']:
                boundary['y_max'] = y

    rectangle_boundary = [boundary['x_max'] - boundary['x_min'], boundary['y_max'] -
boundary['y_min']]

    return np.max(rectangle_boundary) / np.min(rectangle_boundary)


def calc_distance(i, j):
    return np.sqrt(np.abs(i ** 2 - j ** 2))
```

Główny kod programu:

```
if __name__ == '__main__':
    descriptors = [area_descriptor, perimeter_descriptor, roundness_descriptor,
compactness_descriptor, eccentricity_descriptor]
```

```
    images = read_images()
    representatives_numbers = select_representatives_numbers(images)
    train_images, test_images = train_test_split(images, representatives_numbers)

    train_results, test_results = calc_descriptors_for_images(train_images,
test_images)
    results = {}

    for descriptor_name, train_descriptor_results in train_results.items():
        descriptor_results = pd.DataFrame(columns=['predicted', 'real', 'score'])

        for class_iter, test_descriptor_results in
enumerate(test_results[descriptor_name]):
            for test_descriptor_result in test_descriptor_results:
                predicted_class = predict_based_on_distance(test_descriptor_result,
train_descriptor_results)

                row = pd.Series({'predicted': predicted_class, 'real': class_iter,
'score': int(predicted_class == class_iter)})
                descriptor_results = pd.concat([descriptor_results,
row.to_frame().T], ignore_index=True)

        results[descriptor_name] = descriptor_results

    for descriptor_name, descriptor_results in results.items():
        print(f'Descriptor: {descriptor_name}')
        # print(descriptor_results.to_markdown())

        try:
            class_labels = descriptor_results['real'].unique()
            score_by_class = {}

            for class_label in class_labels:
                class_data = descriptor_results[descriptor_results['real'] ==
class_label]
                class_data_count = class_data['score'].count()
                score_by_class[class_label] = class_data['score'].sum() /
class_data_count

            scores_df = pd.DataFrame(np.array([[*score_by_class.keys()],
[*score_by_class.values()]]).T, columns=['class', 'score'])
            scores_df['score'] = scores_df['score'].map(lambda x: f'{(x *
100):0.4f}%')
            print(scores_df.to_markdown())
        finally:
            pass

        dataset_score = descriptor_results["score"].sum() /
descriptor_results.shape[0] * 100
        print(f'Score for whole dataset: {dataset_score:0.4f}%')

        print()
```

Rezultaty

Poprzednie laboratoria

Poniżej wyniki z wielokrotnego uruchamiania programu dla całego zbioru z poprzedniego programu:

```
{'area_descriptor': 0.4, 'perimeter_descriptor': 0.1, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.4, 'perimeter_descriptor': 1.0, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.0, 'perimeter_descriptor': 0.2, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.9, 'perimeter_descriptor': 0.2, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.2, 'perimeter_descriptor': 1.0, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.2, 'perimeter_descriptor': 0.1, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}

{'area_descriptor': 0.7, 'perimeter_descriptor': 1.0, 'roundness_descriptor': 1.0,
'compactness_descriptor': 1.0, 'eccentricity_descriptor': 1.0}
```

Wyniki dla poszczególnych klas wraz z deskryptorami:

area_descriptor	perimeter_descriptor	roundness_descriptor	compactness_descriptor	eccentricity_descriptor
0	0.1	0.3	1	1	1
1	0.3	1	1	1	1
2	0.2	0.1	1	1	1
3	0.6	0.1	1	1	1
4	1	0.1	1	1	1
5	0.2	0	1	1	1
6	0.2	1	1	1	1
7	0.1	0.1	1	1	1
8	0.4	1	1	1	1
9	0.1	0.7	1	1	1

Obecne wyniki

Descriptor: area_descriptor

	class	score
0	1	44.4444%
1	2	22.2222%
2	3	44.4444%
3	4	11.1111%

4	5	44.4444%
5	6	22.2222%
6	7	11.1111%
7	8	55.5556%
8	9	11.1111%
9	10	11.1111%
Score for whole dataset: 27.7778%

Descriptor: perimeter_descriptor
	class	score
0	1	22.2222%
1	2	22.2222%
2	3	88.8889%
3	4	0.0000%
4	5	44.4444%
5	6	11.1111%
6	7	11.1111%
7	8	0.0000%
8	9	0.0000%
9	10	0.0000%
Score for whole dataset: 20.0000%

Descriptor: roundness_descriptor
	class	score
0	1	44.4444%
1	2	22.2222%
2	3	44.4444%
3	4	11.1111%
4	5	44.4444%
5	6	22.2222%
6	7	11.1111%
7	8	55.5556%
8	9	11.1111%
9	10	11.1111%
Score for whole dataset: 27.7778%

Descriptor: compactness_descriptor
	class	score
0	1	44.4444%
1	2	22.2222%
2	3	44.4444%
3	4	11.1111%
4	5	44.4444%
5	6	22.2222%
6	7	11.1111%

7	8	55.5556%
8	9	11.1111%
9	10	11.1111%
Score for whole dataset: 27.7778%

Descriptor: eccentricity_descriptor
	class	score
0	1	22.2222%
1	2	0.0000%
2	3	0.0000%
3	4	0.0000%
4	5	33.3333%
5	6	11.1111%
6	7	0.0000%
7	8	55.5556%
8	9	11.1111%
9	10	22.2222%
Score for whole dataset: 15.5556%

class	area_descriptor	perimeter_descriptor	roundness_descriptor	compactness_descriptor	eccentricity_descriptor	score
1	44.4444%	22.2222%	44.4444%	44.4444%	22.2222%	35.5555%
2	22.2222%	22.2222%	22.2222%	22.2222%	0.0000%	17.7778%
3	44.4444%	88.8889%	44.4444%	44.4444%	0.0000%	44.4444%
4	11.1111%	0.0000%	11.1111%	11.1111%	0.0000%	6.6667%
5	44.4444%	44.4444%	44.4444%	44.4444%	33.3333%	42.2222%
6	22.2222%	11.1111%	22.2222%	22.2222%	11.1111%	17.7778%
7	11.1111%	11.1111%	11.1111%	11.1111%	0.0000%	8.8889%
8	55.5556%	0.0000%	55.5556%	55.5556%	55.5556%	44.4445%
9	11.1111%	0.0000%	11.1111%	11.1111%	11.1111%	8.8889%
10	11.1111%	0.0000%	11.1111%	11.1111%	22.2222%	11.1111%

Scores for all classes by descriptor
area_descriptor 27.7778%
perimeter_descriptor 20.0000%
roundness_descriptor 27.7778%
compactness_descriptor 27.7778%
eccentricity_descriptor 15.5556%

Porównanie

W poprzednich laboratoriach wszystkie obrazy były traktowane jako uczące i testowe. Te laboratoria wymagały wybrania jednego obrazu wzorcowego z każdej klasy, a następnie testowanie na pozostałch.
Wyniki nie są już 100% tak jak w poprzednim przypadku, ponieważ nie testujemy na danych uczących.

Najsłabiej poradziła sobie klasa numer 9 (Tit mouse), a najlepiej klasa 3 (Greator sage grous) oraz 8 (Long eared owl). Pozostałe klasy poradziły sobie różnie: niektóre lepiej, niektóre gorzej.

Patrząc się na łączną dokładność deskryptorów dla wszystkich klas, poradziły sobie one podobnie i żaden z nich nie odstawał znacznie od pozostałych.