Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

FOURIER 2D

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 14 grudnia 2020

Spis treści

1 Cel laboratorium

Celem laboratorium było przeprowadzenie ekstrakcji cech dla logotypów samochodów korzystając z transformaty Fouriera.

2 Wyznaczanie wektora cech

Wektor cech tworzony jest jako wycinek kwadratu o danym boku z widma obrazu. Widmo obrazu obliczane jest przy użyciu funkcji *np.fft.fft2* [1]. Wycięty kwadrat reprezentowany jest

w formie wektora. Kod obliczający podany deskryptor pokazano na listingu 1.

Kod źródłowy 1: Wyznaczanie wektora cech z Fourier 2D
Źródło: Opracowanie własne

```
@staticmethod
def fourier_desc(img, size):
    img_fft = np.fft.fft2(img)
    spectrum = np.log(1 + np.abs(img_fft))
    out = []
    for i in range(0, size):
        tmp = []
        for j in range(0, size):
            tmp.append(spectrum[i][j])
        out.append(tmp)
    return list(np.concatenate(out).flat)
```

3 Klasyfikacja

Klasyfikację przeprowadza się analogicznie względem poprzednich laboratoriów, czyli tworząc słownik wzorców na podstawie obrazów uczących i w procesie predykcji wyszukuje się najbliższy wzorzec za pomocą metryki euklidesowej. Kod prezentujący proces uczenia i predykcji klasyfikatora przedstawiono na listingu 2.

Kod źródłowy 2: Proces uczenia i predykcji klasyfikatora
Źródło: Opracowanie własne

```
def fit(self, X, y):
    self.classes_ = np.unique(y)

    fouriers = []
    labels = []

    for i in range(len(X)):
        im = cv2.imread(X[i], cv2.IMREAD_GRAYSCALE).astype("uint8")
        fourier_desc = self.fourier_desc(im, self.size)
        fouriers.append(fourier_desc)
        labels.append(y[i])

    data = {"fourier": fouriers, "label": labels}

    df = pd.DataFrame.from_dict(data)
    self.template_dict_ = df

def predict(self, X):
    y_pred = []
    for i in range(len(X)):
        x = X[i]
        im = cv2.imread(x, cv2.IMREAD_GRAYSCALE).astype("uint8")
        descriptors = self.fourier_desc(im, self.size)
```

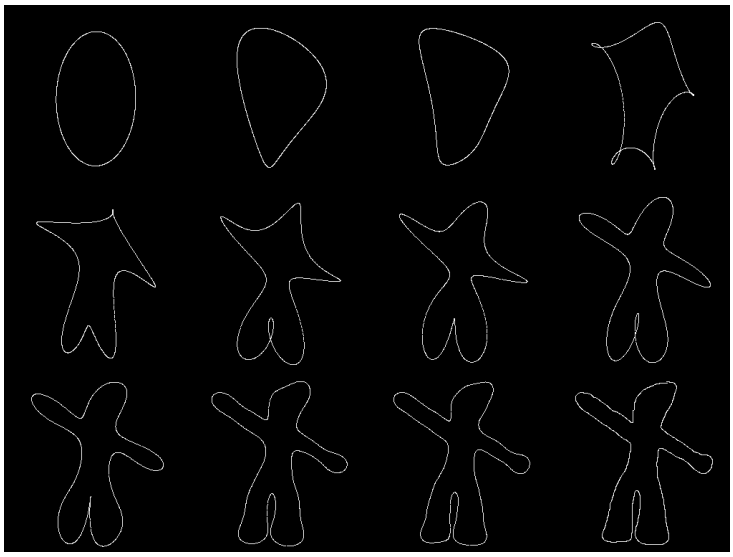
```
            y_pred.append(self.closest_template(descriptors))
        return y_pred

    def closest_template(self, descriptors):
        template_descriptors = self.template_dict_["fourier"].tolist()
        distances = cdist([descriptors], template_descriptors).mean(axis=0)
        closest_label = self.template_dict_.iloc[distances.argmin()]["label"]
        return closest_label
```

4 Eksperymenty

Celem tej części było przebadanie różnych rozmiarów bloku wycinanego z widma obrazu na jakość klasyfikacji.

Wraz ze zwiększaniem liczby komponentów (rozmiaru bloku) zwiększa się dokładność odwzorowania kształtu. Z drugiej strony zmniejszając rozmiar uzyskujemy generalizację kształtu. Takie zniekształcenie można to przedstawić za pomocą przekształcenia odwrotnego (*odwrotnej transformaty fouriera*), po wycięciu danej liczby składowej z widma obrazu. Przedstawiono to na rysunku 1.



Rysunek 1: Rekonstrukcja obrazów po wycięciu bloków róznych rozmiarów z dziedziny widmowej
Po lewej na górze najmniejszy rozmiar bloku, po prawej na dole największy rozmiar bloku
Źródło: http://fourier.eng.hmc.edu/e161/lectures/fd/node1.html

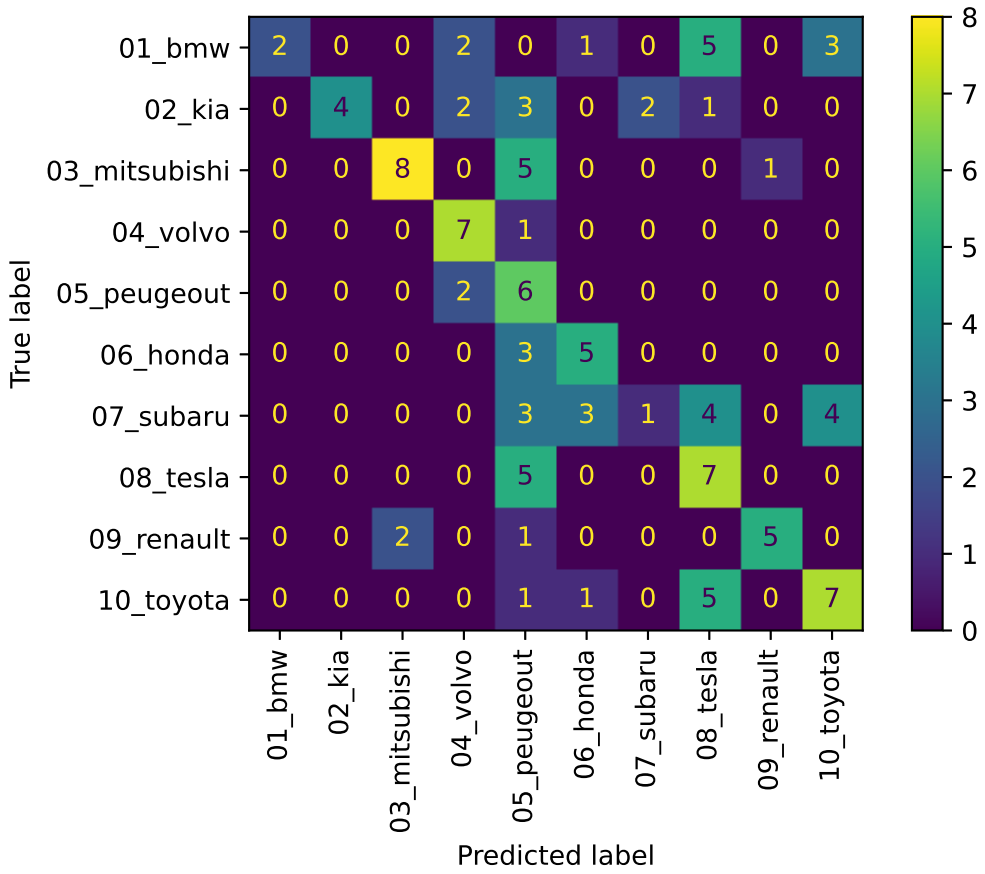
Przeprowadzono badania na 5 różnych rozmiarach bloku, gdzie wartość mówi o długości boku wyciętego kwadratu. Przykładowo dla rozmiaru bloku wynoszącego 5, mówimy o kwadracie 5×5 i uzyskujemy wektor o długości 25.

Badane rozmiary bloku: $5, 20, 30, 35, 40$. Uzyskane wyniki przedstawiono w tabeli 1.

Rozmiar bloku	Dokładność
5	0.401
20	0.428
30	0.464
35	0.446
40	0.428

Tabela 1: Eksperymenty badania rozmiaru bloku w klasyfikacji logotypów samochodu

Najlepszą dokładność, na poziomie 46,4% uzyskano na bloku o wartości 30×30. Wraz z zwiększaniem bloku uzyskujemy utratę dokładności. Wyznaczono macierz konfuzji dla bloku o rozmiarze 30 i przedstawiono go na rysunku (2).



Rysunek 2: Macierz konfuzji dla bloku o rozmiarze 30

5 Podsumowanie

Podczas laboratorium zaimplementowano prosty deskryptor kształtu wykorzystujący transformatę Fouriera.

Zbadano jakość klasyfikacji na podstawie 5 różnych rozmiarów wyciętego bloku z widma obrazu. Uzyskane wyniki przedstawiono w tabeli.

Charakterystyka obiektu badawczego, czyli logotypów samochodów, jest trudna dla deskryptorów kształtu. Jest to spowodowane podobieństwem kształtów pomiędzy logotypami, które przedstawiono w poprzednich sprawozdaniach.

Pewne pary logotypów są trudne do rozpoznania przez człowieka tylko na podstawie kształtu. Są to na przykład logotypu okrągłe lub o kształcie elipsoidy. Z tego powodu wyniki na poziomie 50% są zadowalające.

Bibliografia

[1] Oliphant T. E.: *A guide to numpy*, Trelgol Publishing USA, 2006.

A Całość implementacji Fourier 2D

Kod źródłowy 3: Kod przeprowadzania klasyfikacji za pomocą wycinka z widma obrazu
Źródło: Opracowanie własne

```
import numpy as np
import cv2
import pandas as pd

from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.spatial.distance import cdist


class FourierClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, size=5):
        self.classes_ = None
        self.template_dict_ = None
        self.size = size

    def fit(self, X, y):
        self.classes_ = np.unique(y)

        fouriers = []
        labels = []

        for i in range(len(X)):
            im = cv2.imread(X[i], cv2.IMREAD_GRAYSCALE).astype("uint8")
            fourier_desc = self.fourier_desc(im, self.size)
            fouriers.append(fourier_desc)
            labels.append(y[i])

        data = {"fourier": fouriers, "label": labels}

        df = pd.DataFrame.from_dict(data)
```

```
        self.template_dict_ = df

    def predict(self, X):
        y_pred = []
        for i in range(len(X)):
            x = X[i]
            im = cv2.imread(x, cv2.IMREAD_GRAYSCALE).astype("uint8")
            descriptors = self.fourier_desc(im, self.size)
            y_pred.append(self.closest_template(descriptors))
        return y_pred

    def closest_template(self, descriptors):
        template_descriptors = self.template_dict_["fourier"].tolist()
        distances = cdist([descriptors], template_descriptors).mean(axis=0)
        closest_label = self.template_dict_.iloc[distances.argmin()]["label"]
        return closest_label

    @staticmethod
    def fourier_desc(img, size):
        img_fft = np.fft.fft2(img)
        spectrum = np.log(1 + np.abs(img_fft))
        out = []
        for i in range(0, size):
            tmp = []
            for j in range(0, size):
                tmp.append(spectrum[i][j])
            out.append(tmp)
        return list(np.concatenate(out).flat)


def main():
    from simple_shape_descriptors import prepare_dataset

    X_train, y_train, X_test, y_test = prepare_dataset()

    sizes = [5, 20, 30, 35, 40]

    for size in sizes:
        clf = FourierClassifier(size=size)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        from sklearn.metrics import accuracy_score

        acc = accuracy_score(y_test, y_pred)
        print(f"Size: {size} Acc: {acc}")


def experiment_30():
    from simple_shape_descriptors import prepare_dataset

```

```
    X_train, y_train, X_test, y_test = prepare_dataset()
    size = 30

    clf = FourierClassifier(size=size)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    from sklearn.metrics import accuracy_score

    acc = accuracy_score(y_test, y_pred)
    print(f"Size: {size} Acc: {acc}")

    from sklearn.metrics import plot_confusion_matrix
    import matplotlib.pyplot as plt

    plot_confusion_matrix(clf, X_test, y_test)
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    # main()
    experiment_30()
```