Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wydział
Informatyki

PROSTE DESKRYPTORY KSZTAŁTU

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 6 grudnia 2020

Spis treści

1 Cel laboratorium

Celem laboratorium było przeprowadzenie ekstrakcji cech dla logotypów samochodów korzystając z prostych deskryptorów kształtu oraz porównanie ich dokładności.

2 Wybrane proste deskryptory kształtu

Na laboratorium zaimplementowano następujące deskryptory kształtów:

- pole powierzchni
- długość obwodu
- powłoka wypukła
- średnica okręgu opisanego
- solidność
- zwartość
- kołowość
- mimośród

Przetestowano wybrane deskryptory indywidualnie oraz łącznie, korzystając z podejścia *template matching*, polegającym na wyszukiwaniu najbliższych obiektów. Klasyfikator nauczono na dwóch próbkach uczących.

W implementacji powyższych deskryptorów kształtu wykorzystano bibliotekę *OpenCV* [1] przekazując jako argument obiekt typu *contour* opisujący kontur kształtu.

2.1 Wyznaczanie konturu na podstawie obrazu binarnego

Kontur wyznaczono za pomocą metod dostępnych dzięki bibliotece *OpenCV*.

Kod źródłowy 1: Wyznaczanie konturu
Źródło: Opracowanie własne

```
1  def object_contour(im):
2      ret, thresh = cv2.threshold(im, 127, 255, 0)
3      contours, hierarchy = cv2.findContours(
4          thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE
5      )
6      return contours[1]
```

2.2 Pole powierzchni

Pierwszy prosty deskryptor to pole powierzchni. Jest to liczba pikseli należących do kształtu. Wykorzystano gotową funkcję z biblioteki *OpenCV*.

Kod źródłowy 2: Kod obliczania pola powierzchni
Źródło: Opracowanie własne

```
1 def area(contour):
2     return cv2.contourArea(contour)
```

2.3 Długość obwodu

Kolejny prosty deskryptor to obwód. Jest to suma długości odcinków łączących środki piskeli wchodzących w skład konturu. Przyjmuje się, że element konturu jest kwadratem o boku równym 1.

Kod źródłowy 3: Kod obliczania długości obwodu
Źródło: Opracowanie własne

```
1 def area(contour):
2     return cv2.contourArea(contour)
```

2.4 Powłoka wypukła

Powłoka wypukła (ang.*convex hull*) to obwód najmniejszej figury wypukłej, w którą możemy wpisać badany kształt.

Kod źródłowy 4: Powłoka wypukła
Źródło: Opracowanie własne

```
1 def convex_hull(contour):
2     hull = cv2.convexHull(contour)
3     return hull
```

2.5 Średnica okręgu opisanego

Jest to średnica okręgu opisanego (najmniejszego okręgu, który obejmuje całą figurę).

Kod źródłowy 5: Średnica okręgu opisanego
Źródło: Opracowanie własne

```
1 def diameter(contour):
2     (x, y), radius = cv2.minEnclosingCircle(contour)
3     return radius * 2
```

2.6 Solidność

Solidność (ang. *solidity*) to stosunek pola powierzchni kształtu do pola powierzchni powłoki wypukłej.

Kod źródłowy 6: Solidność
Źródło: Opracowanie własne

```
def solidity(contour):
    # https://docs.opencv.org/3.4/da/dc1/tutorial_js_contour_properties.html
    object_area = area(contour)
    hull = convex_hull(contour)
    hull_area = area(hull)
    solidity = object_area / hull_area
    return solidity
```

2.7 Zwartość

Zwartość (ang. *compactness*) jest to stosunek kwadratu obwodu do pola powierzchni.

$$C = \frac{\text{obwód}^2}{\text{pole}} \tag{1}$$

Kod źródłowy 7: Zwartość
Źródło: Opracowanie własne

```
def compactness(contour):
    object_area = area(contour)
    object_perimeter = perimeter(contour)
    compactness = object_perimeter ** 2 / object_area
    return compactness
```

2.8 Kołowość

Kołowość (ang. *roudness*), inaczej współczynnik kształtu, mówi o tym jak bardzo kształt jest podobny do koła.

$$\gamma = \frac{(\text{obwód})^2}{4\pi(\text{pole})} \tag{2}$$

Kod źródłowy 8: Kołowość
Źródło: Opracowanie własne

```
def roundness(contour):
    p = perimeter(contour)
    a = area(contour)
    gamma = p ** 2 / (4 * np.pi * a)
    return gamma
```

2.9 Mimośród

Mimośród (ang. *eccentricity*) jest to stosunek głównej osi kształtu do osi prostopadłej. Inaczej jest to stosunek dłuższego do krótszego boku minimalnego prostokąta opisanego na figurze.

Kod źródłowy 9: Mimośród
Źródło: Opracowanie własne

```
def eccentricity(contour):
    _, (w, h), _ = cv2.minAreaRect(contour)
    return w / h
```

3 Klasyfikator

Stworzono klasyfikator będący implementacją podejścia *template matching* stosując się do zaleceń biblioteki *scikit learn*. Klasyfikator w procesie uczenia wylicza deskryptory dla podanych obrazów uczących i zapisuje ich wartości w słowniku.

W procesie predykcji próbce testowej przyporządkowywany jest najbliższy wzorzec z przechowywanego słownika zgodnie z metryką euklidesową. Istnieje możliwość obliczania podobieństwa ze względu na jeden deskryptor lub na wektorach będącymi wartościami ze wszystkich deskryptorów. W tym celu zaimplementowano metodą do obliczania deskryptorów dla podanego obrazu oraz wyszukiwania najbliższych wzorców.

Kod źródłowy 10: Proces uczenia
Źródło: Opracowanie własne

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

        areas = []
        perimeters = []
        diameters = []
        solidities = []
        compactnesses = []
        roundnesses = []
        labels = []
        eccentricities = []

        for i in range(len(X)):
            im = cv2.imread(X[i], cv2.IMREAD_GRAYSCALE).astype('uint8')
            contour = ssd.object_contour(im)
            areas.append(ssd.area(contour))
            perimeters.append(ssd.perimeter(contour))
            diameters.append(ssd.diameter(contour))
            solidities.append(ssd.solidity(contour))
            compactnesses.append(ssd.compactness(contour))
            roundnesses.append(ssd.roundness(contour))
            eccentricities.append(ssd.eccentricity(contour))
```

```
            labels.append(y[i])

        data = {'area': areas,
                'perimeter': perimeters,
                'diameter': diameters,
                'solidity': solidities,
                'compactness': compactnesses,
                'roundness': roundnesses,
                'eccentricity': eccentricities,
                'label': labels}
        df = pd.DataFrame.from_dict(data)
        self.template_dict_ = df
```

Kod źródłowy 11: Proces testowania
Źródło: Opracowanie własne

```
    def predict(self, X):
        y_pred = []
        for i in range(len(X)):
            x = X[i]
            descriptors = self.calculate_shape_descriptors(x)
            y_pred.append(self.closest_template(descriptors))
        return y_pred
```

Kod źródłowy 12: Wyznaczanie wartości deskryptorów
Źródło: Opracowanie własne

```
    @staticmethod
    def calculate_shape_descriptors(x):
        im = cv2.imread(x, cv2.IMREAD_GRAYSCALE).astype('uint8')
        contour = ssd.object_contour(im)
        descriptors = [ssd.area(contour), ssd.perimeter(contour),
ssd.diameter(contour), ssd.solidity(contour), ssd.compactness(contour),
                       ssd.roundness(contour), ssd.eccentricity(contour)]
        return descriptors
```

Kod źródłowy 13: Wyznaczanie najbliższego wzorca
Źródło: Opracowanie własne

```
    def closest_template(self, descriptors):
        template_descriptors = self.template_dict_.loc[:, 'area':'eccentricity']
        if self.limited is not None:
            distances = cdist([descriptors[self.limited:self.limited+1]],
template_descriptors.iloc[:, self.limited:self.limited+1]).mean(axis=0)
        else:
            distances = cdist([descriptors], template_descriptors).mean(axis=0)
        closest_label = self.template_dict_.iloc[distances.argmin()]['label']
        return closest_label
```

4 Badania

Przeprowadzono badania testując dokładności poszczególnych deskryptorów. Uzyskano następujące wyniki.

Deskryptor	Dokładność klasyfikacji
Pole powierzchni	28.57%
Obwód	18.75%
Średnica	25.00%
Solidność	43.75%
Zwartość	58.92%
Kołowość	58.92%
Mimośród	69.64%

Tabela 1: Badanie dokładności poszczególnych klasyfikatorów

4.1 Wnioski

Najlepszą dokładność dla badanych logotypów miały deskryptory inwariantne względem skalowania, czyli mimośród, kołowość czy zwartość. Wynika to z faktu, że logotypy w zakresie jednej klasy różnią się głównie skalą obrazka.

Deskryptory, które silnie zależą od rozmiaru kształtu, np. pole powierzchni, obwód czy średnica, dają słabe wyniki dokladność klasyfikacji.

5 Drzewo decyzyjne

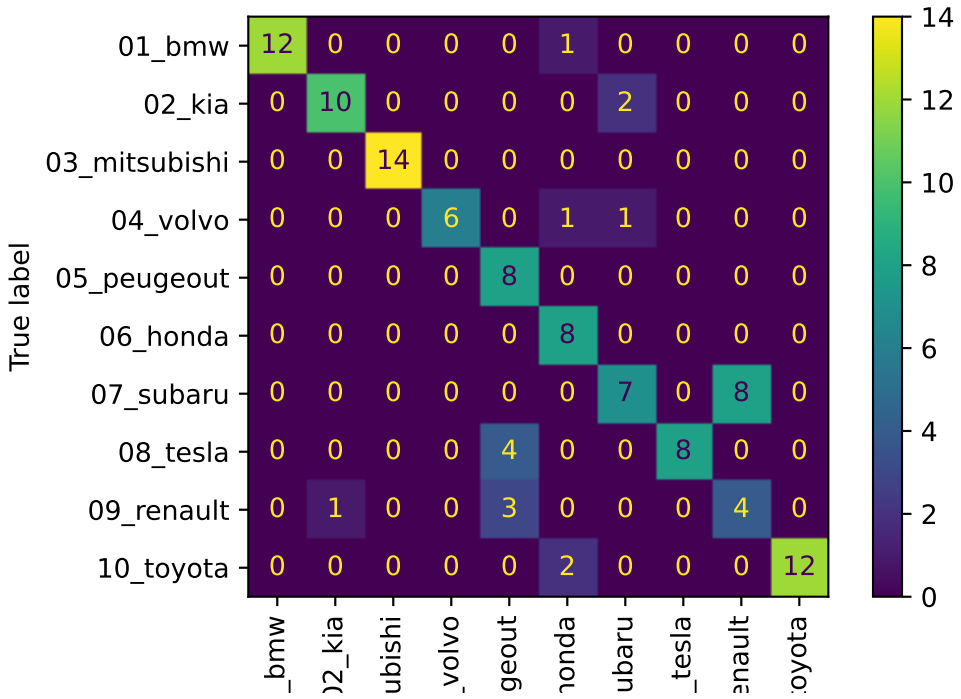
W celu przetestowania dokładności korzystając z zbioru wszystkich cech jako próbek uczących nauczono drzewo decyzyjne na tych samych obrazach co w przypadku poprzedniego badania.

Uzyskano dokładność na poziomie 79% z precyzją 81% i czułością 80%.

Kod źródłowy 14: Wyznaczanie najbliższego wzorca
Źródło: Opracowanie własne

```
1  from sklearn import tree
2  from sklearn.metrics import confusion_matrix
3  from sklearn.metrics import plot_confusion_matrix
4  import matplotlib.pyplot as plt
5
6  X_train_descriptors = [sc.ShapeClassifier.calculate_shape_descriptors(x) for x in
   X_train]
7  X_test_descriptors = [sc.ShapeClassifier.calculate_shape_descriptors(x) for x in
   X_test]
8
9  clf = tree.DecisionTreeClassifier()
10 clf = clf.fit(X_train_descriptors, y_train)
```

```
11
12 y_pred = clf.predict(X_test_descriptors)
13 print(classification_report(y_test, y_pred))
```



Rysunek 1: Macierz konfuzji dla drzewa decyzyjnego