Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wydział
Informatyki

SYGNATURA

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 3 stycznia 2021

Spis treści

1 Cel laboratorium

Celem laboratorium było przeprowadzenie ekstrakcji cech dla logotypów samochodów korzystając z deskryptora sygnatury.

2 Wyznaczanie wektora cech

Wektor cech tworzony jest jako kolejne odległości konturu od wyznaczonego centroida. Elementy konturu są uporządkowane w kolejności zgodnej z wskazówkami zegara. Pierwszy element konturu wyznaczany jest za pomocą przeszukiwania obrazu od góry do dołu, od lewej do prawej strony. Pierwszy piksel będący konturem zostaje elementem startowym sygnatury.

Tak uzyskany wektor odległości może się różnić długością pomiędzy różnymi obiektami. Długość tego wektora zależy od liczby elementów konturu, czyli obwodu danego kształtu. W celu ujednolicenia długości tych wektorów przeprowadzono interpolację i sprowadzono wektory odległości do długości 200 elementów.

Proces wyznaczania sygnatury przedstawiono na rysunku (2).

Kod źródłowy 1: Wyznaczanie wektora cech z sgunatury
Źródło: Opracowanie własne

```
def signature_descriptor(im, length):
    contour = object_contour(im)
    x, y = center_of_contour(im, contour)
    dists = calculate_dists(contour, [x, y])
    downsampled_y, downsampled_x = scale_dists_length(dists, length)

    return downsampled_y, downsampled_x
```

Kod źródłowy 2: Wyznaczanie centroida kształtu
Źródło: Opracowanie własne

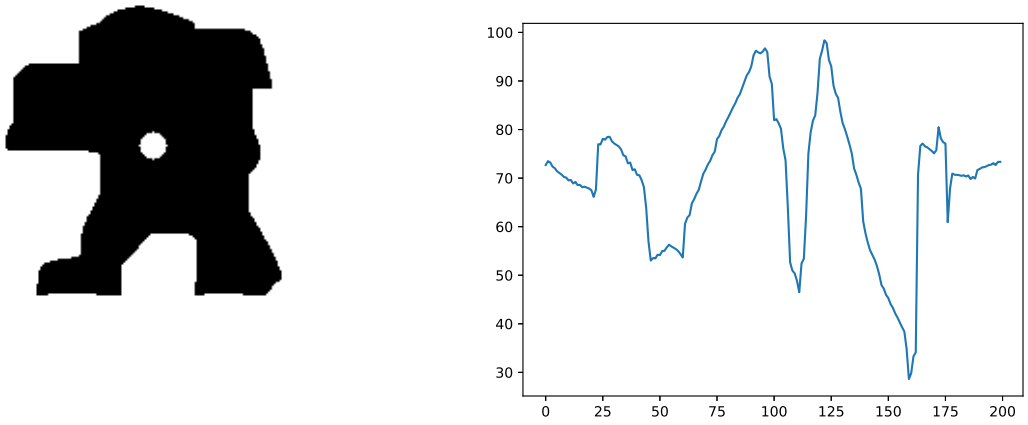
```
def center_of_contour(image, contour):
    M = cv2.moments(contour)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])

    return cX, cY
```

Kod źródłowy 3: Wyliczanie odległości od centroida
Źródło: Opracowanie własne

```
def calculate_dists(contour, center):
    points = []
    for point in contour:
        points.append((point[0, 0], point[0, 1]))

    points = np.array(points)

    dists = distance.cdist([center], points, "euclidean")[0]
    return dists
```

Kod źródłowy 4: Skalowanie wektora (interpolacja)
Źródło: Opracowanie własne

```
def scale_dists_length(dists, length):
    downsampled_x = np.linspace(0, len(dists), length)
    downsampled_y = np.interp(downsampled_x, np.arange(len(dists)), dists)
    return downsampled_y, downsampled_x
```



(a) Kształt z wyznaczonym centroidem

(b) Sygnatura kształtu

Rysunek 1: Proces wyznaczania sygnatury

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 5.

Kod źródłowy 5: Proces uczenia i predykcji klasyfikatora
Źródło: Opracowanie własne

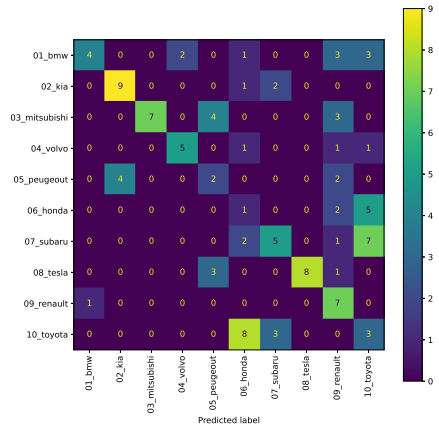
```
class SignatureClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, signature_length=200):
        self.classes_ = None
        self.template_dict_ = None
        self.signature_length = signature_length

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

        signatures = []
        labels = []

        for i in range(len(X)):
            im = cv2.imread(X[i], cv2.IMREAD_GRAYSCALE).astype("uint8")
            signature, _ = signature_descriptor(im, self.signature_length)
            signatures.append(signature)
            labels.append(y[i])

        data = {"signature": signatures, "label": labels}

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

```
23
24      def predict(self, X):
25          y_pred = []
26          for i in range(len(X)):
27              x = X[i]
28              im = cv2.imread(x, cv2.IMREAD_GRAYSCALE).astype("uint8")
29              descriptor, _ = signature_descriptor(im, self.signature_length)
30              y_pred.append(self.closest_template(descriptor))
31          return y_pred
32
33      def closest_template(self, descriptors):
34          template_descriptors = self.template_dict_["signature"].tolist()
35          distances = cdist([descriptors], template_descriptors).mean(axis=0)
36          closest_label = self.template_dict_.iloc[distances.argmin()]["label"]
37          return closest_label
```

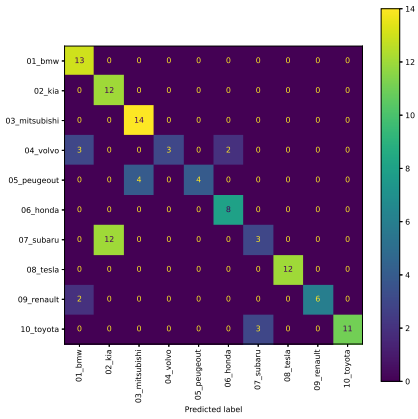
4 Wyniki

Dokładność stworzonego klasyfikatora, działającego na podstawie sygnatury, kształtuje się na poziomie 45%. Na obrazie (2a) przedstawiono macierz konfuzji dla badanego klasyfikatora.

Przetestowano też perceptron wielowarstwowy na takich samych danych wejściowych (wektorze sygnatury). Uzyskano dokładność na poziomie 76,8% dla perceptronu z dwoma ukrytymi warstwami z odpowiednio 100 i 50 neuronami. Macierz konfuzji przedstawiono na obrazie (2b).



(a) Porównanie do wzorca na podstawie odległości euklidesowej



(b) Perceptron wielowarstwowy

Rysunek 2: Porównanie klasyfikatorów dla sygnatury

5 Podsumowanie

Podczas laboratorium zaimplementowano prosty deskryptor kształtu wykorzystujący sygnaturę.

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 40% są zadowalające.