Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

WŁASNY DESKRYPTOR ODCIENI SZAROŚCI - POLAR-DCT GREYSCALE DESCRIPTOR

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 28 stycznia 2021

Spis treści

1 Cel laboratorium

Celem laboratorium było zaproponowanie własnego deskryptora odcieni szarości i przetestowanie go na wybranym zbiorze danych (w moim przypadku były to logotypy samochodów)

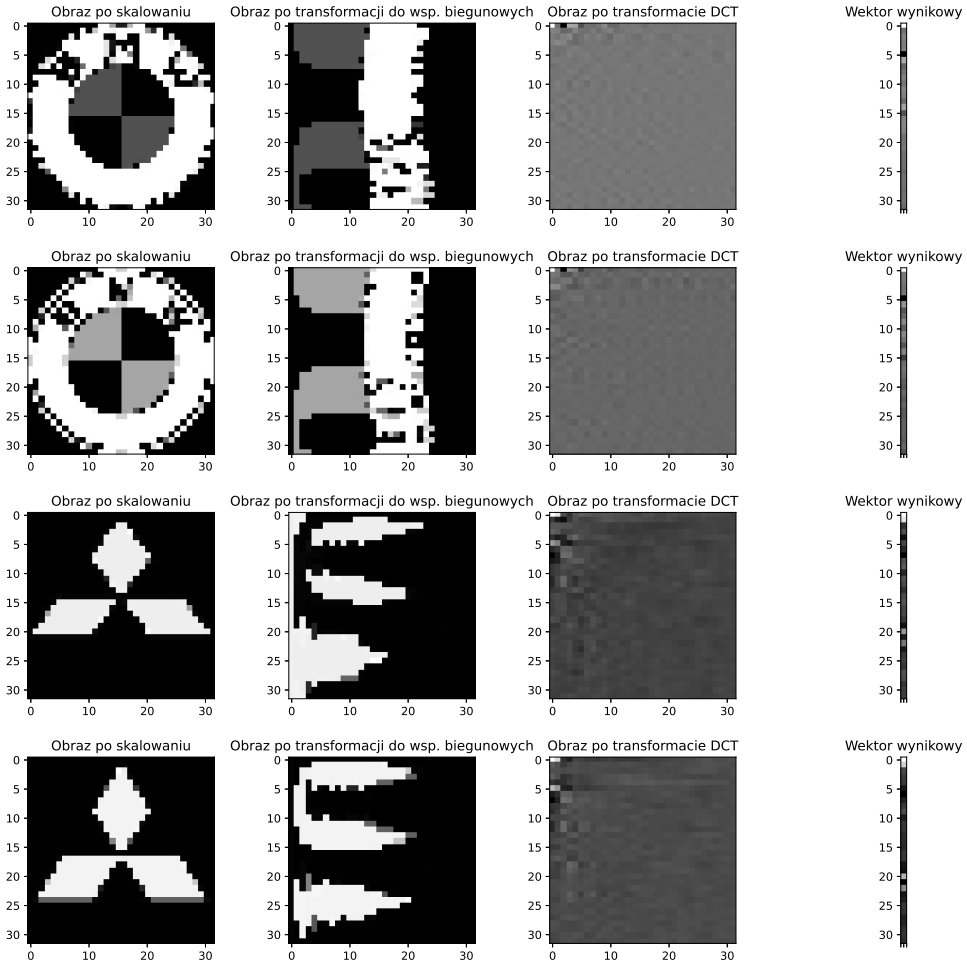
Celem laboratorium była implementacje Polar-Fourier Greyscale Descriptor [1] w uproszczonej formie i przetestowanie go na wybranych zbiorze danych, w moim przypadku były to logotypy samochodów.

2 Zaproponowany deskryptor - Polar-DCT Greyscale Descriptor

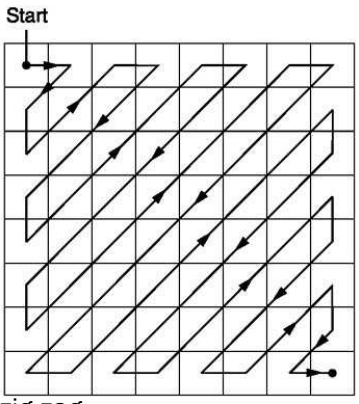
Zaproponowany deskryptor działaniem przypomina Polar-Fourier Greyscale Descriptor [1] w uproszczonej formie.

Metoda składa się z 6 kroków:

1. Transformacja obrazu do odcieni szarości.
2. Skalowanie obrazu do rozmiarów 32×32 pikseli.
3. Wyznaczenie centroida na podstawie punktu ciężkości.
4. Transformacja do współrzędnych biegunowych.
5. Dwuwymiarowa transformata DCT
6. Wybór 32 współczynników DCT za pomocą indeksowania *zig-zag* (rysunek 2)

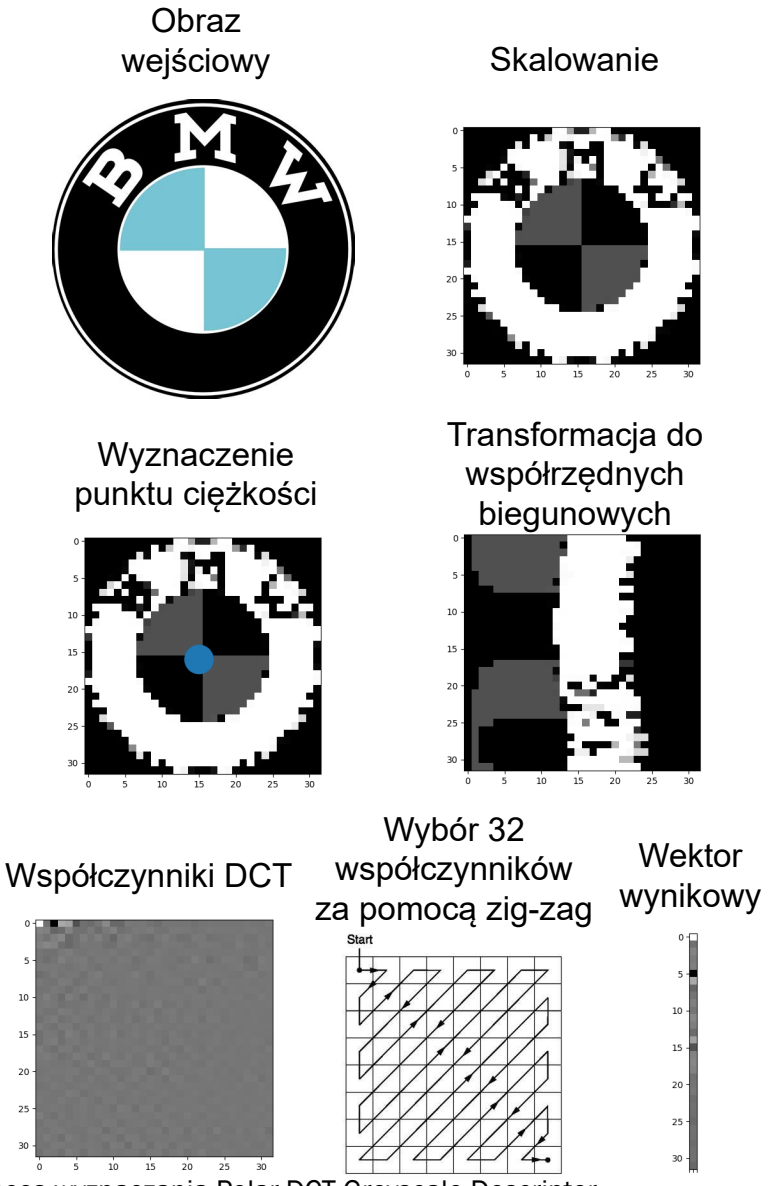


Rysunek 1: Proces wyznaczania cech z logotypów samochodów



Rysunek 2: Sposób indeksowania zig-zag

Źródło: A STUDY ON THE EFFECT OF TRUNCATING THE DISCRETE COSINE TRANSFORM (DCT) COEFFICIENTS FOR IMAGE COMPRESSION [2]



Rysunek 3: Proces wyznaczania Polar-DCT Greyscale Descriptor

Źródło: Opracowanie własne

Kod źródłowy 1: Wyznaczanie Polar-Fourier Greyscale Descriptor
Źródło: Opracowanie własne

```
def my_desc(filename, size):
    # Przekształcenie do skali szarosci
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (32, 32), interpolation=cv2.INTER_LINEAR)
    img = cv2.bitwise_not(img)

    # Przekształcenie do współrzędnych biegunowych obrazu po wektoryzacji
    max_radius = np.sqrt(((img.shape[0] / 2.0) ** 2.0) + ((img.shape[1] / 2.0) **
2.0))
    moment = cv2.moments(img)
    x = int(moment["m10"] / moment["m00"])
    y = int(moment["m01"] / moment["m00"])
    centroid = (x, y)

    polar_image = cv2.linearPolar(
        img, centroid, max_radius, cv2.WARP_FILL_OUTLIERS
    )
    polar_image = polar_image.astype(np.uint8)

    img_dct = dct(dct(polar_image.T, norm='ortho').T, norm='ortho')
    dct_values = zigzag(img_dct)[:32]

    return dct_values


def zigzag(a):
    """
    Create zigzag vector from matrix
    :param a: input matrix
    :returns: vector with length m*n with matrix values under zigzag indexing
    """
    m, n = a.shape
    solution = [[] for i in range(m + n - 1)]

    for i in range(m):
        for j in range(n):
            sum = i + j
            value = a[i][j]
            index = (i, j)
            if sum % 2 == 0:
                solution[sum].insert(0, (value, index))
            else:
                solution[sum].append((value, index))

    output_vector = []
    for i in solution:
        for j in i:
            output_vector.append(j[0])

```

```
49      return output_vector
```

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

```
1  class TemplateClassifier(BaseEstimator, ClassifierMixin):
2      def __init__(self, descriptor, args = {}):
3          self.classes_ = None
4          self.template_dict_ = None
5          self.descriptor_ = descriptor
6          self.args_ = args
7
8      def fit(self, X, y):
9          self.classes_ = np.unique(y)
10
11         features = []
12         labels = []
13
14         for i in range(len(X)):
15             image_path = X[i]
16             feature = self.descriptor_(image_path, **self.args_)
17             features.append(feature)
18             labels.append(y[i])
19
20         data = {"feature": features, "label": labels}
21
22         df = pd.DataFrame.from_dict(data)
23         self.template_dict_ = df
24
25     def predict(self, X):
26         y_pred = []
27         for i in range(len(X)):
28             x = X[i]
29             features = self.descriptor_(x, **self.args_)
30             y_pred.append(self.closest_template(features))
31         return y_pred
32
33     def closest_template(self, descriptors):
34         template_descriptors = self.template_dict_["feature"].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 i wnioski

Przebadane różne długości wektora współczynników DCT

Rozmiar wyciętego bloku	Dokładność klasyfikacji
8	52.67%
16	63.39%
32	74.10%
64	75%
96	75%
128	75%
256	73.214%

Tabela 1: Badanie długości wektora współczynników DCT

Najlepsze wyniki osiągnięto dla wektora o długości 75% osiągając dokładność na poziomie 75%, co pokrywa się z wynikiem Polar-Fourier Greyscale Descriptor ale wymaga dłuższego wektora. Analogiczną dokładność osiąga się w deskryptorze Polar-Fourier Greyscale Descriptor przy 25 elementach współczynników widma.

Porównano zaimplementowany deskryptor do wcześniej omawianych deskryptorów. Wszystkie porównania przeprowadzono na tej samej zasadzie klasyfikacji, czyli porównaniu do wzorca za pomocą metryki euklidesowej. Zaproponowany deskryptor wypada gorzej tylko od UNL-F.

Deskryptor	Dokładność klasyfikacji
Zaproponowany deskryptor	75%
Polar-Fourier Greyscale Descriptor	75%
UNL-F	82.1%
Sygnatura	45%
2D Fourier	46.4%

Tabela 2: Porównanie dokładnośći klasyfikacji dla wybranych deskryptorów

Bibliografia

[1] Frejlichowski D.: Application of the polar–fourier greyscale descriptor to the automatic traffic sign recognition, *International Conference Image Analysis and Recognition*, pp. 506–513, 2015.

[2] Karuppanagounder S., Revathy T., Praveenkumar S., Thiruvenkadam K.: A study on the effect of truncating the discrete cosine transform (dct) coefficients for image compression. *International Journal of Computer Science and Engineering (IJCSE)*, vol. 7, pp. 23–30, lip. 2018.