Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

DESKRYPTORY KOLORU

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 18 stycznia 2021

Spis treści

1 Cel laboratorium

Celem laboratorium było przeprowadzenie ekstrakcji cech dla logotypów samochodów korzystając z prostych deskryptorów koloru.

2 Wykorzystane deskryptory

2.1 Dominant color descriptor

Deskryptor znajduje kolor dominujący za pomocą klasteryzacji kolorów przy użyciu K-Means.

Kod źródłowy 1: Wyznaczanie dominant color descriptor
Źródło: Opracowanie własne

```
def dominant_color_desc(filename):
    # https://gist.github.com/skt7/71044f42f9323daec3aa035cd050884e
    img = cv2.imread(filename)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.reshape((img.shape[0] * img.shape[1], 3))
    kmeans = KMeans(4)
    kmeans.fit(img)
    res = kmeans.cluster_centers_
    res = res.ravel().astype(int)
    return res
```

2.2 Color mean descriptor

Deskryptor wyznacza średni kolor z obrazu.

Kod źródłowy 2: Wyznaczanie dominant color descriptor
Źródło: Opracowanie własne

```
def color_mean_desc(filename):
    img = cv2.imread(filename)
    mean_row = np.average(img, axis=0)
    mean = np.average(mean_row, axis=0)
    return mean
```

2.3 Histogram RGB oraz HSV

Kolejnymi deskryptorami były histogramy w palecie RGB oraz w palecie HSV.

Kod źródłowy 3: Wyznaczanie dominant color descriptor
Źródło: Opracowanie własne

```
def color_hist_rgb_desc(filename):
    """
    Tworzy wektor gdzie kolejno są histogramy 32 elementowe niebieskiego, zielonego
i czerwonego kanału
    """
    img = cv2.imread(filename)
    blue = cv2.calcHist([img], [0], mask=None, histSize=[32], ranges=[0, 256])
    green = cv2.calcHist([img], [1], mask=None, histSize=[32], ranges=[0, 256])
    red = cv2.calcHist([img], [2], mask=None, histSize=[32], ranges=[0, 256])

```

```
    hist = np.ravel(blue).tolist() + np.ravel(green).tolist() +
np.ravel(red).tolist()
    return hist


def color_hist_hsv_desc(filename):
    """
    Histogram w przestrzeni HSV
    """
    img = cv2.imread(filename)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    hue = cv2.calcHist([img], [0], mask=None, histSize=[32], ranges=[0, 256])
    saturation = cv2.calcHist([img], [1], mask=None, histSize=[32], ranges=[0, 256])
    value = cv2.calcHist([img], [2], mask=None, histSize=[32], ranges=[0, 256])

    hist = np.ravel(hue).tolist() + np.ravel(saturation).tolist() +
np.ravel(value).tolist()
    return hist
```

2.4 Color layout descriptor

Deskryptor color layout descriptor pochodzi z standardu MPEG7. Polega na przekształceniu *dct* oraz wektoryzacji za pomocą indeksowania *zigzag*.

Kod źródłowy 4: Wyznaczanie color layout descriptor
Źródło: Opracowanie własne

```
def color_layout_desc(filename, rows=4, cols=4):
    #
https://github.com/tarunlnmiit/irtex-1.0/blob/master/color_layout_descriptor/CLDescriptor.py
    img = cv2.imread(filename)
    img = cv2.resize(img, (128, 128))
    averages = np.zeros((rows, cols, 3))
    imgH, imgW, _ = img.shape
    for row in range(rows):
        for col in range(cols):
            slice = img[imgH // rows * row: imgH // rows * (row + 1),
                    imgW // cols * col: imgW // cols * (col + 1)]
            average_color_per_row = np.mean(slice, axis=0)
            average_color = np.mean(average_color_per_row, axis=0)
            average_color = np.uint8(average_color)
            averages[row][col][0] = average_color[0]
            averages[row][col][1] = average_color[1]
            averages[row][col][2] = average_color[2]
    icon = cv2.cvtColor(
        np.array(averages, dtype=np.uint8), cv2.COLOR_BGR2YCR_CB)
    y, cr, cb = cv2.split(icon)
    dct_y = cv2.dct(np.float64(y))
    dct_cb = cv2.dct(np.float64(cb))
    dct_cr = cv2.dct(np.float64(cr))
```

```
    dct_y_zigzag = []
    dct_cb_zigzag = []
    dct_cr_zigzag = []
    flip = True
    flipped_dct_y = np.fliplr(dct_y)
    flipped_dct_cb = np.fliplr(dct_cb)
    flipped_dct_cr = np.fliplr(dct_cr)
    for i in range(rows + cols - 1):
        k_diag = rows - 1 - i
        diag_y = np.diag(flipped_dct_y, k=k_diag)
        diag_cb = np.diag(flipped_dct_cb, k=k_diag)
        diag_cr = np.diag(flipped_dct_cr, k=k_diag)
        if flip:
            diag_y = diag_y[::-1]
            diag_cb = diag_cb[::-1]
            diag_cr = diag_cr[::-1]
        dct_y_zigzag.append(diag_y)
        dct_cb_zigzag.append(diag_cb)
        dct_cr_zigzag.append(diag_cr)
        flip = not flip
    res = np.concatenate(
        [np.concatenate(dct_y_zigzag), np.concatenate(dct_cb_zigzag),
np.concatenate(dct_cr_zigzag)])
    return res
```

2.5 Scalable color descriptor

Kolejny deskryptor pochodzący z standardu MPEG7. Wykorzystuje przekształcenie falkowe wykorzystując funkcję *haara*.

Kod źródłowy 5: Wyznaczanie color layout descriptor
Źródło: Opracowanie własne

```
def scalable_color_desc(filename):
    image = cv2.imread(filename)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([image], [0], None, [256], [1, 256])
    coeffs = pywt.dwt(hist, 'haar')
    (cA, cD) = coeffs
    res = list(np.concatenate(cA).flat)
    return res
```

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 6.

Kod źródłowy 6: Proces uczenia i predykcji klasyfikatora
Źródło: Opracowanie własne

```
class TemplateClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, descriptor):
        self.classes_ = None
        self.template_dict_ = None
        self.descriptor_ = descriptor

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

        features = []
        labels = []

        for i in range(len(X)):
            image_path = X[i]
            # img = cv2.imread(X[i], cv2.IMREAD_GRAYSCALE).astype("uint8")
            feature = self.descriptor_(image_path)
            features.append(feature)
            labels.append(y[i])

        data = {"feature": features, "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]
            features = self.descriptor_(x)
            y_pred.append(self.closest_template(features))
        return y_pred

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

4 Wyniki

Przebadano zaimplementowane deskryptory koloru na zbiorze logotypów samochodu. Uzyskano następujące wyniki:

Deskryptor	Dokładność
Dominant color descriptor	0.392
Scalable color descriptor	0.312
Color layout descriptor	0.669
Color mean descriptor	0.464
RGB Histogram	0.231
HSV Histogram	0.294

Tabela 1: Porównanie skuteczności klasyfikacji przy użyciu różnych deskryptorów koloru.

5 Podsumowanie

Podczas laboratorium zaimplementowano proste deskryptory koloru. Uzyskano zadowalające wyniki osiągające około 50% dokładności.

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

Pewne pary logotypów są trudne do rozpoznania przez człowieka tylko na podstawie koloru. Są to na przykład logotypy całkowicie czarne. Z tego powodu wyniki na poziomie 66% dla deskryptora *color layour descriptor* są zadowalające.