Sprawozdanie

Przetwarzanie Języka Naturalnego

Temat projektu:

System do odczytywania hieroglifów przy użyciu scikit-image.

Piotr Jaśkowski

Cel i opis projektu

Celem projektu było wykorzystanie biblioteki sikit-image do utworzenia programu służącego do odczytywanie hieroglifów ze zdjęć.

Wykorzystane technologie

Scikit-image - to biblioteka open-source do przetwarzania obrazów dla języka programowania Python. Zawiera algorytmy do segmentacji, przekształceń geometrycznych, manipulacji przestrzenią kolorów, analizy, filtrowania, morfologii, wykrywania cech i wiele innych.

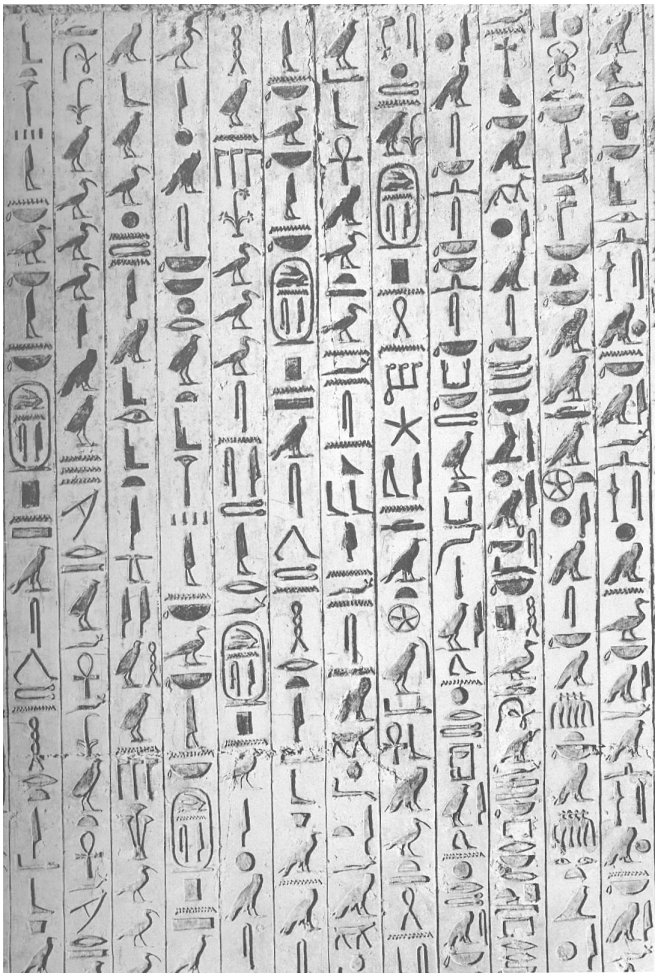
Tkinter - jest biblioteką Pythona służąca do tworzenia GUI programu. Tkinter jest dostępny na większości platform uniksowych, w tym macOS, a także na systemach Windows.

matplotlib - to biblioteka do tworzenia wykresów dla języka programowania Python i jego rozszerzenia NumPy.

numpy - jest biblioteką obsługującą duże, wielowymiarowe tablice i macierze, wraz z dużym zbiorem wysokopoziomowych funkcji matematycznych do operowania na tych tablicach.

3. Wykorzystane dane

Dane wykorzystane w projekcie pochodzą ze strony: https://github.com/fgimbert/Hieroglyphs/tree/master/hieroglyphs są to wycięte z zdjęć obrazki hieroglifów(przykład poniżej).



Łącznie około 4000 hieroglifów. Kolejnymi danymi jest lista unicodów hieroglifów pochodząca ze strony(poniżej przykłady):
http://www.alanwood.net/unicode/egyptian-hieroglyphs.htm

77824	\u00013000 A001	EGYPTIAN HIEROGLYPH
77825	\u00013001 A002	EGYPTIAN HIEROGLYPH

3.1 Przetwarzanie danych

Większość z obrazków była już podpisana te które nie były zostały odrzucone. Obrazki różniły się od siebie tym jak są zapisane to znaczy miały różny kształty to znaczy zawierały różne ilości informacji.

```
In [8]:  1 col[0].shape
Out[8]: (77, 34)
```

```
In [12]:  1 col[999].shape
Out[12]: (28, 64, 3)
```

Poniższy fragment kodu pokazuje jak te problemy zostały rozwiązane:

In [3]:
```
def fix_shape(img):
    if img.shape[2] == 3:
        new_img = np.delete(img, np.arange(0, img.size, 3))
        new_img = np.delete(new_img, np.arange(0, new_img.size, 2))
        new_img.shape = (img.shape[0], img.shape[1])
    elif img.shape[2]  == 4:
        new_img = np.delete(img, np.arange(0, img.size, 2))
        new_img = np.delete(new_img, np.arange(1, new_img.size, 2))
        new_img.shape = (img.shape[0], img.shape[1])
    return new_img
```

In [5]:
```
data = []
label = []
for i in range(len(col)):
    name = col.files[i].split('_')[1].split('.')[0]
    if name != 'UNKNOWN':
        label.append(name)
        if(col[i].ndim > 2):
            data.append(fix_shape(col[i]))
        else:
            data.append(col[i])
```

Z tych obrazków które miały dodatkowe informacje zostały one wyrzucone i kształ zmieniony do reszty obrazków, z nazw obrazków zostały wyciągnięte interesujące mnie informacje czyli nazwa hieroglifów. Nazwy które zostały wyciągnięte różniły się od nazw które byłe wykorzystane do w unicodach, np dla unicode wyglądało to tak: A001 a dla nazw z obrazków A1, czyli istniały dodatkowe zera.

Poniższy kod wczytał plik txt z unicodami i wyciągną z nich interesujące nas informacje i przeformatowuje nazwę tak aby pasowała do tych z obrazków.

In [2]:
```
unicode = []
glyph_id = []

with open("unicode.txt") as file:
    lines = file.readlines()
for i in range(len(lines)):
    glyph_id.append(lines[i][42] + lines[i][44:46].lstrip('0'))
    unicode.append(lines[i][0])

```

4. Wyszukiwanie hieroglifów

Do wyszukiwanie hieroglifów została wykorzystana funkcja match_template z biblioteki scikit-image. Jest to funkcja która wykorzystuje szybka znormalizowaną korelację krzyżową do znajdowania szablonów na obrazie. Jest to nie perfekcyjne rozwiązanie do tego problemu, ale chyba jedyne z tej biblioteki. Za pomocą tej funkcji sprawdzane po kolei wszystkie hieroglify, po czym są one sortowane od najbardziej pasujących do najmniej, fragment kodu odpowiedzialny za to poniżej.

```
    score = []
    xs = []
    ys = []

    for i in range(len(data)):
        if log == True:
            j = (i + 1) / len(data)
            sys.stdout.write('\r')
            sys.stdout.write("[%-20s] %d%%" % ('='*int(20*j), 100*j))
            sys.stdout.flush()

        if np.any(np.less(img.shape, data[i].shape)):
            if log == True:
                sys.stdout.write("size conflic skippping hieroglyph \n")
        else:
            result = match_template(img, data[i])
            ij = np.unravel_index(np.argmax(result), result.shape)
            x, y = ij[::-1]
            score.append(result[y,x])
            xs.append(x)
            ys.append(y)

    best_fit = heapq.nlargest(4000, range(len(score)), key=score.__getitem__)
```

Mając już listę najbardziej pasujących hieroglifów wyciągane są z nich interesujące nas dane czyli lokalizacja i ich nazwa. Każdy kolejny najbardziej pasujących hieroglif jest sprawdzany czy znajduje sie w lokalizacji zajętej już przez wcześniejszy hieroglif, bez tego nakładałyby się one jeden na drugi. Ilość szukanych hieroglifów, (w kodzie poniżej zmienna ‘number’) oraz to jak blisko siebie mogą się znajdować (w kodzie poniżej zmienna overlap_tolerance) są definiowane przez użytkownika.

```
    uniqe_best_fit = []
    loc_labels = []
    loc_x = []
    loc_y = []
    j = 0
    while len(loc_labels) < number:
        conflict = check_conflict(xs[best_fit[j]], ys[best_fit[j]],
                        loc_x, loc_y, overlap_tolerance)
        if(conflict == True):
            j = j + 1
        else:
            loc_labels.append(label[best_fit[j]])
            loc_x.append(xs[best_fit[j]])
            loc_y.append(ys[best_fit[j]])
            uniqe_best_fit.append(best_fit[j])
            j = j + 1
```

-

```
def check_conflict(x, y, x_list, y_list, overlap_tolerance):
    x_conflict = False
    y_conflict = False
    for xs in x_list:
        if x in range(xs-overlap_tolerance, xs+overlap_tolerance):
            x_conflict = True
    for ys in y_list:
        if y in range(ys-overlap_tolerance, ys+overlap_tolerance):
            y_conflict = True
    if((x_conflict == True) and
       (y_conflict == True)):
        return True
    else:
        return False
```

Mając już listę wybranych hieroglifów są one rysowane wraz ze swoja nazwą na orginalnym obrazku i zwracane. Lista wybranych hieroglifów wraz z ich unicodami jest również wypisywan lub zwracana do menu w wersji z gui.

```
    fig = plt.figure(figsize=(50, 20))
    ax = plt.subplot(1, 3, 2)
    ax.imshow(img, cmap=plt.cm.gray)

    for i in uniqe_best_fit:
        rect = plt.Rectangle((xs[i], ys[i]), data[i].shape[1], data[i].shape[0],
                             edgecolor='r', facecolor='none')
        name = plt.text(xs[i]+5, ys[i]+5, label[i], color='r', size = 'x-large')
        ax.add_patch(rect)

    plt.show()

    print("Zanlieziono następujące hierogliphy: ")
    loc_unicodes = []
    for k in loc_labels:
        result = glyph_id.index(k)

        print(k + ' - ' + unicode[result])

```

5. Wnioski

Program wykonuje swoje założenia to znaczy robi to co miał robić jednak wyszukiwanie hieroglifów zajmuje minutę lub dwie. Lepszym rozwiązaniem byłoby wykorzystanie API tensorflow do wyszukiwania obiektów ni jeżli biblioteki scikit-image.

6. Źródła

- baza danych: https://github.com/fgimbert/Hieroglyphs/tree/master/hieroglyphs
- unicodes: http://www.alanwood.net/unicode/egyptian-hieroglyphs.html
- sklearn: https://scikit-image.org/docs/stable/index.html
    - https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_template.html