**UNIVERSITATEA DIN BUCUREȘTI**
**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

# Concepte și Aplicatii în Vederea Artificială

## Tema 2

## Detectarea și recunoașterea facială a personajelor din serialul de desene animate Familia Flinstone

**Iancu Florentina-Mihaela**

**CTI, grupa 461**

# Cuprins

**1. Introducere.....3**

**2. Task 1. .....4**

**3. Task 2. .....6**

**4. Rezultate .....7**

# Introducere

Proiectul a fost creat folosind un server Jupyter și Python 3 în PyCharm.

În dezvoltarea acestui program am folosit următoarele librării: os, numpy (as np), cv2 (as cv), matplotlib.pyplot (as plt), LinearSVC from sklearn.svm, deepcopy from copy, hog from skimage.feature, glob, pickle, ntpath, timeit.

Pentru a obține imaginile de antrenament din cele originale, am folosit următorul cod:

```
file_barney = open('antrenare/wilma_annotations.txt', 'r')
Lines = file_barney.readlines()

for idx, line in enumerate(Lines):
    temp = line.split(' ')
    img = cv.imread('antrenare/wilma/'+temp[0])
    xmin = int(temp[1])
    ymin = int(temp[2])
    xmax = int(temp[3])
    ymax = int(temp[4])
    y = ymax-ymin
    x = xmax-xmin
    if(y > x):
        cropped_image = img[ymin:ymax, xmin-(y-x)//2:xmax+(y-x)//2]
    else:
        cropped_image = img[ymin-(x-y)//2:ymax+(x-y)//2, xmin:xmax]
    try:
        resized_image = cv.resize(cropped_image, (85, 85))
        cv.imwrite('antrenare/exemple_pozitive/wilma_'+str(idx)+'.jpg', resized_image)
    except Exception as e:
        print(str(e))
```

Toate path-urile care trebuie schimbate pentru rularea corectă a codului sunt în Parameters la începutul codului împreună cu un set de comentarii despre cum trebuie alterate.

```
class Parameters:
    def __init__(self):
        self.base_dir = '../Proiect2'
        # path-urile acestea devin redundante daca nu trebuie sa generam modelul
        self.dir_pos_examples = os.path.join(self.base_dir,
'antrenare/exemple_pozitive')
        self.dir_neg_examples = os.path.join(self.base_dir,
'antrenare/exemple_negative')

        # path-ul pentru imaginire de testare
        self.dir_test_examples = os.path.join(self.base_dir,'validare/validare')

        # path-ul pentru rezultatele corecte ale imaginilor de testare
        self.path_annotations = os.path.join(self.base_dir,
'validare/validare_annotations.txt')

        # path-ul pentru folder-ul cu modelele deja generate
        self.dir_save_files = os.path.join(self.base_dir, 'salveazaFisiere')
        if not os.path.exists(self.dir_save_files):
            os.makedirs(self.dir_save_files)
            print('directory created: {} '.format(self.dir_save_files))
        else:
            print('directory {} exists '.format(self.dir_save_files))
```

# Task 1.

Pentru setup, am decis că dimensiunea exemplelor pozitive o să fie 85 x 85 de pixeli. Având între 100 și 500 de poze pentru exemplele negative (în funcție de situație), am ajuns la concluzia că folosind 4.000 / 6.000 de patch-uri pentru descriptorii negativi se obțin rezultate mai bune decât dacă se folosesc mai puțin de 2.000.

```
# set the parameters
        self.dim_window = 85  # exemplele pozitive (fete de oameni cropate) au 85x85
pixeli
        self.dim_hog_cell = 6  # dimensiunea celulei
        self.dim_descriptor_cell = 85  # dimensiunea descriptorului unei celule
        self.overlap = 0.5
        self.number_positive_examples = 6881  # numarul exemplelor pozitive
        self.number_negative_examples = 4000  # numarul exemplelor negative
        self.overlap = 0.5
        self.has_annotations = False
        self.threshold = 0
```

În rezolvarea task-ului, am folosit materialele de la laboratoarele 9 și 10, precum și descriptorii _hog. Majoritatea modificărilor au fost făcute asupra clasei *Parameters*, asupra funcției *run()* din clasa *FacialDetector* și asupra celulelor unde se rulează efectiv codul.

La bucățile de cod deja existente din funcția *run()*, am făcut câteva modificări la sliding window. Pentru a mă apropia de un rezultat mai bun, am creat vectorul *scale_nr* care conține coeficienții de modificare ai mărimii pozei. Aceste numere sunt folosite asupra pozei pentru a o mări sau micșora, dar și asupra coordonatelor obținute (*x_min, y_min, x_max, y_max*) pentru a le potrivi pe imaginea originală.

Ajustarea coordonatelor nu se face prin înmulțire, ci se face împărțindu-le la *dim*.

```
img = cv.imread(test_files[i], cv.IMREAD_GRAYSCALE)

scale_nr = {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, 1.25, 1.5}
image_scores = []
image_detections = []

for dim in scale_nr:
    img_sliding_win = cv.resize(img, None, fx = dim, fy = dim)

    hog_descriptors = hog(img_sliding_win, pixels_per_cell=(self.params.dim_hog_cell,
self.params.dim_hog_cell),
                          cells_per_block=(2, 2), feature_vector=False)
    num_cols = img_sliding_win.shape[1] // self.params.dim_hog_cell - 1
    num_rows = img_sliding_win.shape[0] // self.params.dim_hog_cell - 1
    num_cell_in_template = self.params.dim_window // self.params.dim_hog_cell - 1

    for y in range(0, num_rows - num_cell_in_template):
        for x in range(0, num_cols - num_cell_in_template):
            descr = hog_descriptors[y:y + num_cell_in_template, x:x +
num_cell_in_template].flatten()
            score = np.dot(descr, w)[0] + bias

            if score > self.params.threshold:
                x_min = int(x * self.params.dim_hog_cell / dim)
                y_min = int(y * self.params.dim_hog_cell / dim)
```

```
                x_max = int((x * self.params.dim_hog_cell + self.params.dim_window)/
dim)
                y_max = int((y * self.params.dim_hog_cell + self.params.dim_window)/
dim)
                image_detections.append([x_min, y_min, x_max, y_max])
                image_scores.append(score)
```

Pentru a salva rezultatele în fișiere .npy, am folosit funcția *save()* din numpy. Am împărțit rezultate în funcție de task și am creat fișierele folosind datele deja generate în cod: *detections, file_names, scores.*

```
def output_file(detections, scores, file_names, add, params):
    if add.strip() == 'all_faces':
        path = os.path.join(params.dir_result,'task1')
    else:
        path = os.path.join(params.dir_result,'task2')

    file = os.path.join(path, 'detections_'+add.strip()+'.npy')
    np.save(file, detections)

    file = os.path.join(path, 'file_names_'+add.strip()+'.npy')
    np.save(file, file_names)

    file = os.path.join(path, 'scores_'+add.strip()+'.npy')
    np.save(file, scores)
```

Această funcție output_file() este folosită la finalul codului.

```
# output files
output_file(detections, scores, file_names, char.strip(), params)
```

## Task 2.

Diferența majoră dintre cele două task-uri este modul de rulare a codului. Folosim un vector de string-uri cu numele caracterelor *characters* și iterăm prin ele, făcând modificări unde este nevoie: numărul de exemple pozitive (care trebuie să fie exact numărul de imagini folosite), câteva fișiere folosite pentru antrenarea modelului sau pentru verificarea rezultatelor.

```
characters = ['fred','barney','betty','wilma']

for char in characters:
    file_number =
glob.glob('../Proiect2/antrenare/antrenare_'+char.strip()+'/*')

    params.number_positive_examples = len(file_number) # numarul exemplelor
pozitive
    params.number_negative_examples = 6000 # numarul exemplelor negative

    params.dir_pos_examples = 'antrenare/antrenare_'+char.strip()
    params.dir_neg_examples = 'antrenare/exemple_negative_task2'
    params.path_annotations =
'validare/task2_'+char.strip()+'_gt_validare.txt'
```

Există câteva modificări dinamice la salvarea și accesarea modelelor și a descriptorilor:

```
negative_features_path = os.path.join(params.dir_save_files,
'descriptoriExempleNegative_'+char.strip()+ str(params.dim_hog_cell) + '_' +
                        str(params.number_negative_examples) + '.npy')
```

```
positive_features_path = os.path.join(params.dir_save_files,
'descriptoriExemplePozitive_'+char.strip()+'_'+ str(params.dim_hog_cell) + '_' +
                        str(params.number_positive_examples) + '.npy')
```

```
def train_classifier(self, training_examples, train_labels, char):
    svm_file_name = os.path.join(self.params.dir_save_files, 'best_model_%d_%d_%d_%s'
%
                                 (self.params.dim_hog_cell,
self.params.number_negative_examples,
                                  self.params.number_positive_examples, char))
```

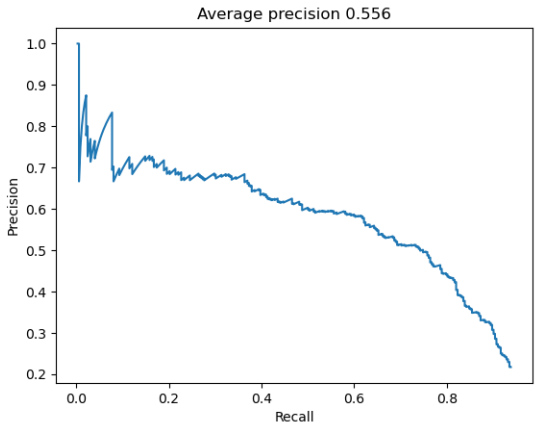
În cazul folosirii funțiilor care generează graficele *precizie_medie*, fiecare variantă v-a crea propria imagine.

```
def eval_detections(self, detections, scores, file_names, fig =
'precizie_medie.png'):
    …
    plt.savefig(os.path.join(self.params.dir_save_files, fig))
```

# Rezultate

Folosind codul pe care l-am explicat mai sus, am obținut următoarele rezultate:

Task 1.




Task 2. (Fred, Barney, Wilma, Betty)