Importação das bibliotecas utilizadas

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage import feature
import os
import glob
from sklearn.svm import SVC, NuSVC, LinearSVC
from skimage.filters import sobel
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn import preprocessing
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Constantes

```
#GABOR
THETA = 32
SIGMA = (1,2)
KSIZE = 8
#LBP
LBP_NUM = 8;
LBP_RADIUS = 1
# LBP_EPS = 1e-7

IMG_SIZE = 128
BLOCKS_ROWS = 32
BLOCKS_COLS = 32

#Número de imagens para carregar por classe
NUM_IMAGES = 100

RAW_FEATURE_VEC_SIZE = 9

BASE_DIR_DATASET = "D:\dev\Tiago\simpsons_dataset_reduce_"
```

Funções

Para gabor são gerados N vetores correspondente as variações de direção THETA e os comprimentos de onda SIGMA. O resultado do vetor é compactado usando a função **createVector** que reduz o vetor para 3 dimensões. Logo o tamanho total do vetor final de gabor g é: $n(g) = (\theta * n(\sigma) * 3) * n(c)$

onde $n(x)$ é o tamanho de um conjunto x e c os canais da imagem

Em python pode ser calculado como: THETA * len(SIGMA) * 3 * NUM_CANAIS (1 para imagens em escala de cinza e 3 para coloridas)

```
 size_gabor = THETA * len(SIGMA) * 3 * 3
```

```
def createGaborFeatures(img):
    ksize= KSIZE
    df = pd.DataFrame()
    num = 0;
    #Itera sobre as variáveis theta e sigma para gerar vetor de filtros
    for theta in range(THETA):
        theta = theta / 4. * np.pi
        for sigma in SIGMA:  #Sigma
            num += 1;
            lamda = np.pi/4
            gamma = 0.2
            gabor_label = 'Gabor' + str(num)
            kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma, 0, ktype=cv2.CV_

            #aplica o filtro recem gerado através de convolução na imagem
            fimg = np.asarray(cv2.filter2D(img, cv2.CV_8UC3, kernel))
            feature_vector = np.zeros(9, dtype=np.float32)
            #Por praticidade, utiliza um Dataframe Pandas
            df['Gabor_'+str(num)] = createVector(fimg)

    return df
```

Calcula LPB para a imagem, o vetor final é o histograma do padrão

O Tamanho do vetor é o tamanho do histogram que é igual a LBP_NUM+2

```
size_lbp = LBP_NUM+2
```

```
def createLBPHistogram(gray):
    df = pd.DataFrame()
    #Gera um·histograma·uniforme·sobre·o·resultado·como·vetor·de·caractarísticas
    lbp = feature.local_binary_pattern(gray, LBP_NUM, LBP_RADIUS, method="uniform")
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, LBP_NUM+3), range=(0, LBP_NUM+2))
    df["LBP"] = hist
    return df
```

Sobel

No caso do sobel, utiliza-se a média, mediana e variância dos pixels resultantes como vetor, desta forma sobel contribui com 3 posições para o vetor final.

```
size_sobel = 3
```

```
def createSobelFeature(gray):
    df = pd.DataFrame()

    res_sobel = sobel(gray)

    feature_vector = []
    feature_vector.append(res_sobel.mean())
    feature_vector.append(res_sobel.var())
    feature_vector.append(np.median(res_sobel))

    df["Sobel"] = np.asarray(feature_vector)
    return df
```

Função para, dado um vetor de características, compacta o mesmo criando retornando apenas a média, mediana e variância do vetor inteiro, para cada canal da imagem

Vetor de pixels da imagem

Esse vetor é composto de valores extraídos diretamente da imagem sem sofrer nenhum processamento prévio por filtros. A imagem é dividida em uma grade de tamanho onde cada quadrante tem BLOCKS_ROWS x BLOCKS_ROWS de dimensões, então é extraído um vetor comprimido **createVector** para cada quadrante.

O tamanho, por tanto é: (w / BLOCKS_COLS) * (h / BLOCKS_ROWS) * 3 * 3 = (w / BLOCKS_COLS) * (h / BLOCKS_ROWS) * 9

onde w é a largura da imagem e h a altura em pixels

```
#Cria um vetor sobre informações dos pixeel da imagem
def generateRawFeatures(img):
    blocks = reshapeBlock(img, BLOCKS_ROWS, BLOCKS_COLS)
    df = pd.DataFrame()

    index = 0
    for b in blocks:
        df["Raw_"+str(index)] = createVector(b)
        index += 1
    return df


def createVector(img, asPandaDF = False):
    feature_vector = None

    if( len(img.shape) > 2):
        feature_vector = np.zeros(9)
        feature_vector[0] = np.mean(img[:,:,0])
        feature_vector[1] = np.mean(img[:,:,1])
        feature_vector[2] = np.mean(img[:,:,2])

        feature_vector[3] = np.var(img[:,:,0])
```

```
        feature_vector[4] = np.var(img[:,:,1])
        feature_vector[5] = np.var(img[:,:,2])

        feature_vector[6] = np.median(img[:,:,0])
        feature_vector[7] = np.median(img[:,:,1])
        feature_vector[8] = np.median(img[:,:,2])
    else:
        feature_vector = np.zeros(3)
        feature_vector[0] = np.mean(img)
        feature_vector[1] = np.var(img)
        feature_vector[2] = np.median(img)

    res = None
    if asPandaDF:
        df = pd.DataFrame()
        df["Raw"] = np.asarray(feature_vector)
        res = df
    else:

        res = feature_vector
    return res



size_raw = (IMG_SIZE//BLOCKS_COLS) * (IMG_SIZE//BLOCKS_COLS) * 9
```

Após todas declarada todas as funções de *features* temos o tamanho total do vetor:

```
print("Tamanho total do vetor de características => {}".format(size_gabor+size_lbp+size_sobel+size_r
```

```
 Tamanho total do vetor de características => 733
```

```
#Função para facilitar a normalização, representando um dado vetor entre 0 e 1
def normalizeArray(arr):
    return preprocessing.minmax_scale(arr, feature_range=(0,1))


#Cria uma Grid, dividindo a imagem em nrows por ncols
def reshapeBlock(arr, nrows, ncols):
    if len(arr.shape) == 3:
        h, w, _ = arr.shape
        assert h % nrows == 0, f"{h} rows is not evenly divisible by {nrows}"
        assert w % ncols == 0, f"{w} cols is not evenly divisible by {ncols}"
        return (arr.reshape(h//nrows, nrows, -1, ncols,3)
                   .swapaxes(1,2)
                   .reshape(-1, nrows, ncols,3))

    else:
        h, w = arr.shape
        assert h % nrows == 0, f"{h} rows is not evenly divisible by {nrows}"
        assert w % ncols == 0, f"{w} cols is not evenly divisible by {ncols}"

        return (arr.reshape(h//nrows, nrows, -1, ncols)
```

```
                    .swapaxes(1,2)
                    .reshape(-1, nrows, ncols))
def loadImageFeatures(path, img = None):
    if type(img) == type(None):
        img = cv2.imread(path)
        #Redimensionando a imagem para um formato quadrado menor
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    #Converte para escalas de cinza

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    gabor = createGaborFeatures(img)

    sobel = createSobelFeature(img)

    lbp = createLBPHistogram(gray)
    lbp = normalizeArray(lbp)

    raw = generateRawFeatures(img/255)

    gabor1D = np.expand_dims(gabor, axis=0)
    gabor1D = normalizeArray(gabor1D.reshape(-1))

    #Montagem do vetor final
    f_vector = np.empty(0)

    f_vector = np.append(f_vector,sobel)
    f_vector = np.append(f_vector,gabor1D)
    f_vector = np.append(f_vector,lbp)
    f_vector = np.append(f_vector,raw)


    df = pd.DataFrame()
    df["features"] = f_vector;

    return df
```

Itera sobre o diretório para obter informações das classes e arquivos

```
classes = []
vectors = []

# Procura por diretórios dentro do diretório raiz da base escolhida
# cada sub-diretório será uma classe
for dirs in glob.glob(BASE_DIR_DATASET+"/*"):
    curr_label = dirs.split("\\")[-1]
    ind = 0

    for imgp in glob.glob(dirs+"\*.jpg")[0:NUM_IMAGES]:
        ind += 1
        progress = round(ind / NUM_IMAGES * 100)
        print("Carregando classe {}...{}%       ".format(curr_label,progress ),end='\r')
        classes.append(curr_label)
        #Carrega o vetor de características da imagem e adiciona a base
```

```
        f = loadImageFeatures(imgp)
        vectors.append(np.asarray(f).ravel())
    print("\n")

# Converte tudo para matriz Numpy
print("Pronto!")
vectors = np.asarray(vectors)
classes = np.asarray(classes)
```

```
     Carregando classe bart_simpson...100%

     Carregando classe homer_simpson...100%

     Carregando classe lisa_simpson...100%

     Carregando classe maggie_simpson...100%

     Carregando classe marge_simpson...100%

     Pronto!
```

Criação do modelo

Primeiro separa-se o conjunto em conjunto treino e conjunto teste

```
from sklearn.preprocessing import LabelEncoder
le = preprocessing.LabelEncoder()
le.fit(list(set(classes)))

X_train, X_test, y_train, y_test = train_test_split(vectors, le.transform(classes), test_size=0.35,
```

Inicializa uma instância de um Algoritmo de Aprendizado

```
model = RandomForestClassifier(random_state=321)
# model = SVC(C=1000, random_state=321, max_iter=99000, break_ties=True)

#Treina o modelo
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
print("Score: %f " % (score * 100))
```

```
     Score: 74.857143
```

Avaliação do modelo

```
#Avaliação
y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
```

```
acc = accuracy_score(y_test, y_pred)
rec = recall_score(y_test, y_pred,average='macro')
print("Accuracy: {0:.4g}%".format(acc*100))
print("Recall: {0:.4g}%".format(rec*100))
```

```
Accuracy: 74.86%
Recall: 75.44%
```

Exporta *features* para arquivo csv

```
pd_classes = pd.DataFrame()
pd_classes["Classes"] = classes
pd_values = pd.DataFrame(np.around(vectors, decimals=12))
final = pd.concat((pd_classes,pd_values), axis=1)
final.to_csv("simpsons_db.csv", index=False)
```

Utiliza um exemplo aleatório

```
#Toda vez que for executada esse célula será escolhido uma classe e imagem aleatória
import random
lc = list(set(classes))
cl = random.choice(lc)
ind = random.randrange(101,120)
pth = os.path.join(BASE_DIR_DATASET,cl,"pic_{}.jpg".format(f'{ind:04d}'))
feat = np.asarray(loadImageFeatures(pth)).reshape(1,-1)
pred = model.predict(feat)
pred_s = str(le.inverse_transform([pred[0]])[0])
res_s = lambda T: "Correto" if T == cl else "Errado"

print("[{}] {}! - Classe real: {} - predito: {}".format(ind, res_s(pred_s),cl, pred_s ))
```

```
[107] Errado! - Classe real: maggie_simpson - predito: lisa_simpson
```

✓ 0s conclusão: 10:44