IMAGE PROCESSING

Identification of different types of fonts contained in binary images using descriptors extracted separately and also with Histogram of Oriented Gradient (HOG).

Use of K-means and Random Forest models for prediction.

The development presented below has purposes linked to academic research. Because I am using Google Colab, some codes are specific to the platform.

I was not allowed to leave the dataset available with this document.

Some objects have names written in Portuguese, as my teacher asked for at the time.

- IMG011 = LETTER A
- IMG012 = LETTER B
- IMG013 = LETTER C

In [ ]:

```
# Unzip the train set
!unzip train_set
```

In [ ]:

```
# Libraries
import os
import cv2
import pandas as pd
import numpy as np
import glob
from matplotlib import pyplot as plt
from google.colab import files
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from skimage.feature import hog
from skimage import data, exposure

# Model evaluation metrics
from sklearn.metrics import classification_report, \
accuracy_score, f1_score, confusion_matrix

%matplotlib inline
plt.rcParams["figure.figsize"] = [5,5]
```

In [ ]:

```
# Empty lists for training and test data
x_treino = []
y_treino = []

x_teste = []
y_teste = []

# Final files to export
final_treino = pd.DataFrame ()
final_treino["IMAGEM"] = ()
final_treino["ANOTACAO"] = ()

final_teste = pd.DataFrame ()
final_teste["IMAGEM"] = ()
final_teste["ANOTACAO"] = ()
```

TRAIN SET

In [ ]:

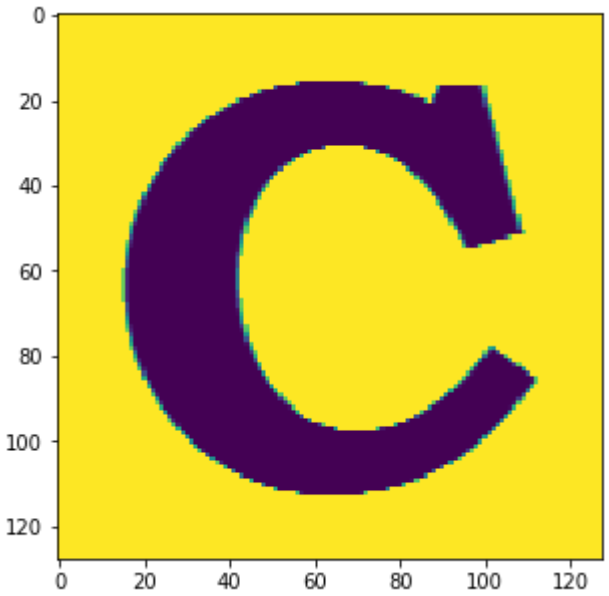
```
# Inserting all the images in x_train and naming each one by the name of the
# folder in y_train.

# Also inserting the information in the final_train for export
fil = glob.glob ("/content/*/*.png")
for myFile in fil:
    image = cv2.imread (myFile, 0)
    x_treino.append (image) # Adding the image in x
    y_treino.append (myFile[9:10]) # labeling in y

    # Adding the image name to the final file
    final_treino.loc[myFile, 'IMAGEM'] = myFile[11:-4]

    # Adding the label name to the final file
    final_treino.loc[myFile, 'ANOTACAO'] = myFile[9:10]

# Analyzing the x_train size
print('x_treino shape:', np.array(x_treino).shape)
```

```
x_treino shape: (2397, 128, 128)
```

In [ ]:

```
# Verification
plt.imshow(x_treino[1500])
print(y_treino[1500])
```

C



In [ ]:

```
# Reset the final file index
final_treino = final_treino.reset_index(drop=True)
```

In [ ]:

```
# Validation in the final file
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO
0	img012-00236	B
1	img012-00584	B
2	img012-00406	B
3	img012-00472	B
4	img012-00554	B

STARTING TREATMENTS IN THE TRAINING SET

The first treatment we will do is to invert the colors so that it is possible \ to identify the letters in white and the background in black. \ For this, we will use the "NOT" method that will make the negative of the images.

In [ ]:

```
# Creating a dataset to allocate the modified images
x_treino_not = []
```

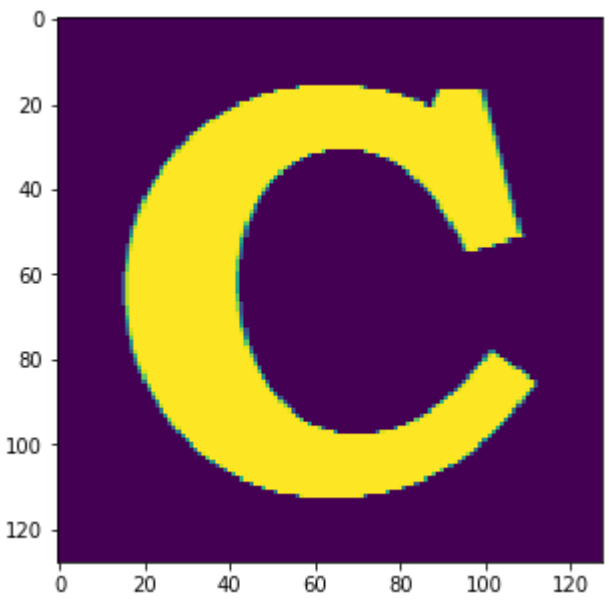
In [ ]:

```
# Modifing the colors
for image in x_treino:
  img = cv2.bitwise_not(image)
  x_treino_not.append(img)
```

In [ ]:

```
# Verification
plt.imshow(x_treino_not[1500])
print(y_treino[1500])
```

C



skeleton of Zhang-Suen

We will add the Zhang-Suen skeleton treatment.

I used this method because we have many fonts designed in a complex way. \ Thus, the Zhang-Suen skeleton is a good solution.

I chose to work with skeletons to mitigate variations in the thickness of different sources

In [ ]:

```
def neighbours(image, x, y):
    "Return 8-neighbours of image point P1(x,y), in a clockwise order"
    img = image
    x_1, y_1, x1, y1 = x-1, y-1, x+1, y+1

    neigh = [img[x_1][y], img[x_1][y1], img[x][y1], img[x1][y1],
             img[x1][y], img[x1][y_1], img[x][y_1], img[x_1][y_1]]

    return neigh
```

In [ ]:

```
def transitions(neighbours):
    "No. of 0,1 patterns (transitions from 0 to 1) in the ordered sequence"
    n = neighbours + neighbours[0:1]
    return sum( (n1, n2) == (0, 1) for n1, n2 in zip(n, n[1:]) )
```

In [ ]:

```
def zhangSuen(image):
    "the Zhang-Suen Thinning Algorithm"
    skeleton = image.copy()  # deepcopy to protect the original image
    skeleton = cv2.divide(skeleton, 255)
    changing1 = changing2 = 1        #  the points to be removed (set as 0)

    #  iterates until no further changes occur in the image
    while changing1 or changing2:

        # Step 1
        changing1 = []
        rows, columns = skeleton.shape   # x for rows, y for columns
        for x in range(1, rows - 1):     # No. of  rows
            for y in range(1, columns - 1): # No. of columns
                P2, P3, P4, P5, P6, P7, P8, P9 = n = neighbours(skeleton, x, y)

                # Condition 0: Point P1 in the object regions
                if (skeleton[x][y] == 1     and
                    2 <= sum(n) <= 6   and    # Condition 1: 2<= N(P1) <= 6
                    transitions(n) == 1 and    # Condition 2: S(P1)=1
                    P2 * P4 * P6 == 0  and    # Condition 3
                    P4 * P6 * P8 == 0): # Condition 4
                    changing1.append((x,y))
        for x, y in changing1:
            skeleton[x][y] = 0

        # Step 2
        changing2 = []
        for x in range(1, rows - 1):
            for y in range(1, columns - 1):
                P2,P3,P4,P5,P6,P7,P8,P9 = n = neighbours(skeleton, x, y)
                if (skeleton[x][y] == 1   and        # Condition 0
                    2 <= sum(n) <= 6  and       # Condition 1
                    transitions(n) == 1 and      # Condition 2
                    P2 * P4 * P8 == 0 and       # Condition 3
                    P2 * P6 * P8 == 0): # Condition 4
                    changing2.append((x,y))
        for x, y in changing2:
            skeleton[x][y] = 0

    skeleton = cv2.multiply(skeleton, 255)
    return skeleton
```

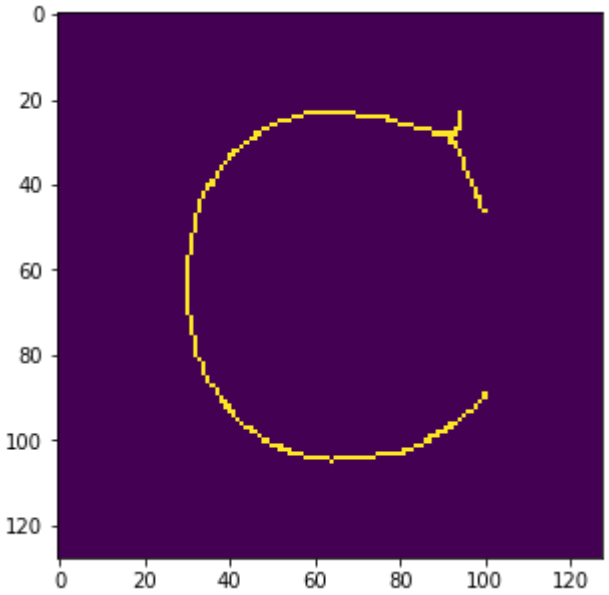
In [ ]:

```
# Dataset with the skeleton
x_treino_esq = []
count = 0

for image in x_treino_not:
  img_skel = image
  _, img_skel = cv2.threshold(img_skel, 127, 255, cv2.THRESH_OTSU)
  skeleton_new = zhangSuen(img_skel)
  x_treino_esq.append(skeleton_new)
  print(count)
  count += 1
```

In [ ]:

```
# Skeleton verification
plt.imshow(x_treino_esq[1500])
print(y_treino[1500])
```

C



ADDING THE DESCRIPTORS TO THE TRAINING SET

Applying the image area

In [ ]:

```
# Creating the file to insert the areas
base_area = []
for img in x_treino_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + \
                                 cv2.THRESH_OTSU)
  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  area = cv2.contourArea(biggest)
  base_area.append(area)
```

In [ ]:

```
# Crossing area information with existing information
final_treino = final_treino.join(pd.DataFrame(base_area\
                                              )).rename(columns={0:'AREA_OBJ'})
```

Applying the image height

In [ ]:

```
# Creating the file to insert the heights
base_altura = []
for alt in x_treino_esq:
  x, y, w, h = cv2.boundingRect(alt)
  altura = h
  base_altura.append(altura)
```

In [ ]:

```
# Crossing height information with existing information
final_treino = final_treino.join(pd.DataFrame(base_altura\
                                              )).rename(columns={0:'ALTURA'})
```

Applying the image width

In [ ]:

```
# Creating the file to insert the widths
base_largura = []
for lar in x_treino_esq:
  x, y, w, h = cv2.boundingRect(lar)
  largura = w
  base_largura.append(largura)
```

In [ ]:

```
# Crossing width information with existing information
final_treino = final_treino.join(pd.DataFrame(base_largura\
                                              )).rename(columns={0:'LARGURA'})
```

Applying the contour area

In [ ]:

```
# Creating the file to insert the areas
base_area2 = []
for area in x_treino_esq:
  x, y, w, h = cv2.boundingRect(area)
  objeto = w * h
  base_area2.append(objeto)
```

In [ ]:

```
# Crossing area information with existing information
final_treino = final_treino.join(pd.DataFrame(base_area2\
                                              )).rename(columns={0:'AREA_CONTORNO'})
```

Applying horizontal balancing

In [ ]:

```
# Creating the file to insert horizontal balancing
balan_hor = []
for hor in x_treino_esq:
  x, y, w, h = cv2.boundingRect(hor)
  horizontal = (64-(w/2))
  balan_hor.append(horizontal)
```

In [ ]:

```
# Crossing horizontal balancing information with existing information
final_treino = final_treino.join(pd.DataFrame(balan_hor\
                                              )).rename(columns={0:'BALAN_HOR'})
```

Applying vertical balancing

In [ ]:

```
# Creating the file to insert vertical balancing
balan_ver = []
for ver in x_treino_esq:
  x, y, w, h = cv2.boundingRect(ver)
  vertical = (64-(h/2))
  balan_ver.append(vertical)
```

In [ ]:

```
# Crossing vertical balancing information with existing information
final_treino = final_treino.join(pd.DataFrame(balan_ver\
                                              )).rename(columns={0:'BALAN_VER'})
```

Applying the ratio of images

In [ ]:

```
# Creating the file to insert ratio
base_razao = []
for raz in x_treino_esq:
  x, y, w, h = cv2.boundingRect(raz)
  razao = h/w
  base_razao.append(razao)
```

In [ ]:

```
# Crossing ratio information with existing information
final_treino = final_treino.join(pd.DataFrame(base_razao\
                                              )).rename(columns={0:'RAZAO'})
```

Applying image compactness

In [ ]:

```
# Creating the file to insert compactness
base_compacidade = []

for img in x_treino_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  area = cv2.contourArea(biggest)
  perimetro = cv2.arcLength(biggest, True)
  compacidade = ((perimetro**2)/area) if area != 0 else 0
  base_compacidade.append(compacidade)
```

In [ ]:

```
# Crossing compactness with existing information
final_treino = final_treino.join(pd.DataFrame(base_compacidade\
                                              )).rename(columns={0:'COMPACIDADE'})
```

Applying the contour perimeter

In [ ]:

```
# Creating the file to insert contour perimeter
base_perimetro = []

for img in x_treino_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  perimetro = cv2.arcLength(biggest, True)
  base_perimetro.append(perimetro)
```

In [ ]:

```
# Crossing contour perimeter with existing information
final_treino = final_treino.join(pd.DataFrame(base_perimetro\
                                              )).rename(columns={0:'PERIM_CONTORNO'})
```

Applying rectangle

In [ ]:

```
# As we already have the area, width and height values calculated in the
# table, we will do the calculation directly
final_treino['RETANG.'] = final_treino['AREA_OBJ'] / (final_treino['ALTURA'] *\
                                                      final_treino['LARGURA'])
```

Applying the Euler number

In [ ]:

```
# Creating the file to insert Euler number
base_euler = []

for img in x_treino_esq:
  contornos,_ = cv2.findContours(img,cv2.RETR_CCOMP,cv2.CHAIN_APPROX_SIMPLE)

  # We take 2 out of the outline because 1 represents the letter itself and
  # the other 1 is from the formula.
  euler = 2 - len(contornos)
  base_euler.append(euler)
```

In [ ]:

```
# Crossing Euler number with existing information
final_treino = final_treino.join(pd.DataFrame(base_euler\
                                              )).rename(columns={0:'EULER'})
```

Applying the area and perimeter of the convex closure

In [ ]:

```
# Creating the file to insert the areas
base_area3 = []
base_perimetro2 = []

for img in x_treino_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  hull = cv2.convexHull(biggest)
  area = cv2.contourArea(hull)
  base_area3.append(area)
  perimetro = cv2.arcLength(hull, True)
  base_perimetro2.append(perimetro)
```

In [ ]:

```
# Crossing the area and perimeter information of the closure
# with the existing ones
final_treino = final_treino.join(pd.DataFrame(base_area3\
                                              )).rename(columns={0:'AREA_FECHO'})
final_treino = final_treino.join(pd.DataFrame(base_perimetro2\
                                              )).rename(columns={0:'PERIM_FECHO'})
```

Applying convexity

In [ ]:

```
# As we already have the closing and contour perimeter values in the table,
# we will do the calculation directly
final_treino['CONVEXIDADE'] = final_treino['PERIM_FECHO'] / final_treino['PERIM_CONTORN
O']
```

Applying solidity

In [ ]:

```
# As we already have the closure and contour area values in the table,
# we will do the calculation directly
final_treino['SOLIDEZ'] = final_treino['AREA_CONTORNO'] / final_treino['AREA_FECHO']
```

In [ ]:

```
# Validation
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img012-00236	B	4358.5	89	78	6942	25.0
1	img012-00584	B	3284.0	89	63	5607	32.5
2	img012-00406	B	3826.0	81	57	4617	35.5
3	img012-00472	B	4358.5	89	78	6942	25.0
4	img012-00554	B	3612.0	93	64	5952	32.0

Normalizing the dataset

In [ ]:

```
# Creating the scale
scaler = StandardScaler()
scaler.fit(final_treino.iloc[:,3:17])
```

Out[ ]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [ ]:

```
# Scaling the train_set
final_treino_scaled = pd.DataFrame(scaler.transform(final_treino.iloc[:,3:17]))
final_treino_scaled.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	
0	0.303837	0.439304	0.576152	-0.439304	-0.303837	-0.339693	-0.341409	-0.740824	1.2
1	0.303837	-0.649192	-0.490560	0.649192	-0.303837	0.377600	-0.341098	-1.012867	1.1
2	-0.755505	-1.084591	-1.281604	1.084591	0.755505	0.399657	-0.341866	-1.346063	2.0
3	0.303837	0.439304	0.576152	-0.439304	-0.303837	-0.339693	-0.341409	-0.740824	1.2
4	0.833509	-0.576626	-0.214893	0.576626	-0.833509	0.484337	-0.340337	-0.346762	1.2

Applying K-means in training set

In [ ]:

```
# Creating the model and fitting the scaled dataset
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, \
                n_init=10, random_state=0)

kmeans.fit(final_treino_scaled)
```

Out[ ]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

In [ ]:

```
# Predict and already aggregating in the final file
final_treino['pred_y_kmeans'] = kmeans.predict(final_treino_scaled)
```

In [ ]:

```
# Validation
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img012-00236	B	4358.5	89	78	6942	25.0
1	img012-00584	B	3284.0	89	63	5607	32.5
2	img012-00406	B	3826.0	81	57	4617	35.5
3	img012-00472	B	4358.5	89	78	6942	25.0
4	img012-00554	B	3612.0	93	64	5952	32.0

Applying Random Forest in train_set

In [ ]:

```
# Creating the RF model and fitting the scaled base with the label being the
# column: "ANOTACAO" of the final file
model_rf = RandomForestClassifier()
model_rf.fit(final_treino_scaled, final_treino['ANOTACAO'])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Fut
ureWarning: The default value of n_estimators will change from 10 in versi
on 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
                       max_depth=None, max_features='auto', max_leaf_nodes
=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
# Predict and already aggregating in the final file
final_treino['pred_y_rf'] = model_rf.predict(final_treino_scaled)
```

In [ ]:

```
# Validation
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img012-00236	B	4358.5	89	78	6942	25.0
1	img012-00584	B	3284.0	89	63	5607	32.5
2	img012-00406	B	3826.0	81	57	4617	35.5
3	img012-00472	B	4358.5	89	78	6942	25.0
4	img012-00554	B	3612.0	93	64	5952	32.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by Kmeans
pd.crosstab(final_treino["ANOTACAO"],final_treino["pred_y_kmeans"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_kmeans	0	1	2
ANOTACAO			
A	62.327910	13.016270	24.655820
B	45.556946	6.257822	48.185232
C	63.829787	4.881101	31.289111

In [ ]:

```
# Cross-validation between the label and the Random Forest
pd.crosstab(final_treino["ANOTACAO"],final_treino["pred_y_rf"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_rf	A	B	C
ANOTACAO			
A	99.874844	0.000000	0.125156
B	0.125156	99.874844	0.000000
C	0.000000	0.000000	100.000000

We can already analyze that the tree model performed much better \ than K-means with the default parameters. We will apply on the test_set, \ and also compare with the HOG to analyze and the performances.

Criando a base HOG (treino) e aplicando o K-means e random forest na mesma

In [ ]:

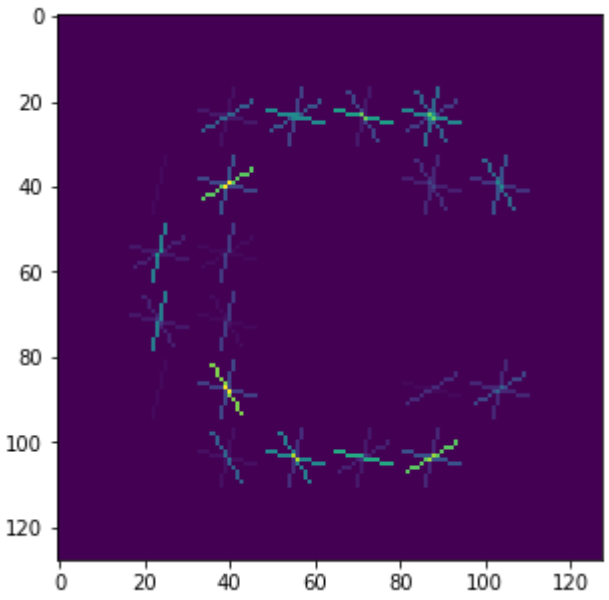
```
# HOG
base_hog = []

for img in x_treino_esq:
  fd, hog_image = hog(img, orientations=8, pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1), visualize=True, multichannel=False)
  base_hog.append(hog_image)
```

In [ ]:

```
# Viewing an image with HOG
plt.imshow(base_hog[1500])
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f4a4a7e0438>
```



In [ ]:

```
# Adjusting the dimensions of the dataset to apply k-means
base_hog_ok = np.array(base_hog)
nsamples, nx, ny = base_hog_ok.shape
base_hog_final = base_hog_ok.reshape((nsamples,nx*ny))
```

In [ ]:

```
# Building the models on the HOG file

# K-means
kmeans_hog = KMeans(n_clusters=3, init='k-means++', max_iter=300, \
                    n_init=10, random_state=0)

kmeans_hog.fit(base_hog_final)

final_treino['pred_y_kmeans_hog'] = kmeans_hog.predict(base_hog_final)
```

In [ ]:

```
# Validation
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img012-00236	B	4358.5	89	78	6942	25.0
1	img012-00584	B	3284.0	89	63	5607	32.5
2	img012-00406	B	3826.0	81	57	4617	35.5
3	img012-00472	B	4358.5	89	78	6942	25.0
4	img012-00554	B	3612.0	93	64	5952	32.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by Kmeans_hog
pd.crosstab(final_treino["ANOTACAO"],final_treino["pred_y_kmeans_hog"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_kmeans_hog	0	1	2
ANOTACAO			
A	98.748436	1.251564	0.000000
B	13.391740	1.627034	84.981227
C	6.758448	92.740926	0.500626

As we can see, when we apply the HOG, k-means considerably improves \ its assertiveness, when compared to the application made in the \ previous descriptors.

In [ ]:

```
# Creating the RF model and fitting the HOG file with the label being the
# column: "ANOTACAO" in the final file
model_rf_hog = RandomForestClassifier()
model_rf_hog.fit(base_hog_final, final_treino['ANOTACAO'])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Fut
ureWarning: The default value of n_estimators will change from 10 in versi
on 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
                       max_depth=None, max_features='auto', max_leaf_nodes
=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
# Predict in the HOG dataset and already adding the column at the end
final_treino['pred_y_rf_hog'] = model_rf_hog.predict(base_hog_final)
```

In [ ]:

```
# Validation
final_treino.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img012-00236	B	4358.5	89	78	6942	25.0
1	img012-00584	B	3284.0	89	63	5607	32.5
2	img012-00406	B	3826.0	81	57	4617	35.5
3	img012-00472	B	4358.5	89	78	6942	25.0
4	img012-00554	B	3612.0	93	64	5952	32.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by RF_hog
pd.crosstab(final_treino["ANOTACAO"],final_treino["pred_y_rf_hog"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_rf_hog	A	B	C
ANOTACAO			
A	100.0	0.000000	0.000000
B	0.0	100.000000	0.000000
C	0.0	0.125156	99.874844

As we can see, the HOG + Random Forest model achieves 100% assertiveness \ in training. We will now do all the applications for the test_set

TEST SET

In [ ]:

```
# Unzip the train set
!unzip teste_set
```

In [ ]:

```
# Inserting all images in dataset x
# Also inserting the information in the final dataset
fil = glob.glob ("/content/*.png")
for myFile in fil:
    image = cv2.imread (myFile, 0)
    x_teste.append (image) # Aggregating the image on x

    # Adding the image name to the final file
    final_teste.loc[myFile, 'IMAGEM'] = myFile[9:-4]

    # Adding the image name (6 first digits)
    final_teste.loc[myFile, 'ANOTACAO'] = myFile[9:15]

# Analisando o tamando da base
print('x_teste shape:', np.array(x_teste).shape)
```

```
x_teste shape: (615, 128, 128)
```

In [ ]:

```
# Resetting the final file index
final_teste = final_teste.reset_index(drop=True)
```

STARTING TREATMENTS IN THE TEST SET

In [ ]:

```
# Creating a file to allocate the changed images
x_teste_not = []
```

In [ ]:

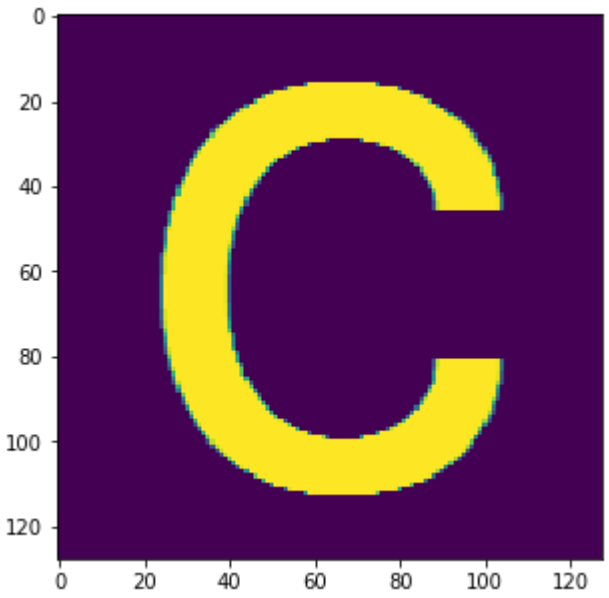
```
# Changing the image and background colors
for image in x_teste:
  img = cv2.bitwise_not(image)
  x_teste_not.append(img)
```

In [ ]:

```
# Verification
plt.imshow(x_teste_not[150])
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f4a4a762ef0>
```



Applying skeleton

In [ ]:

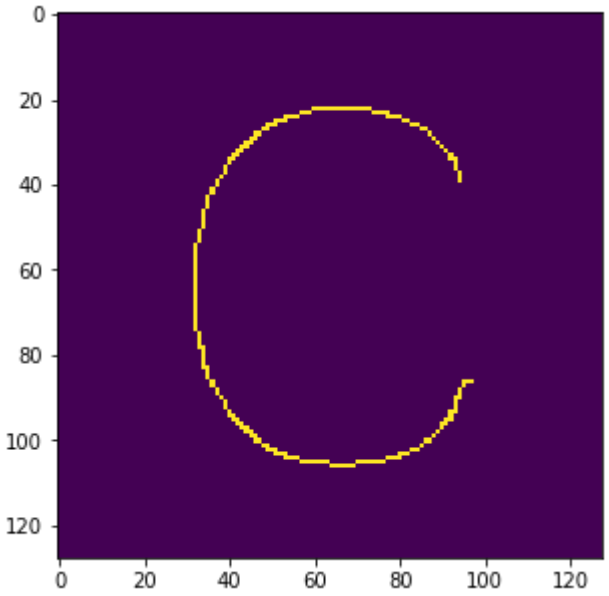
```
#Dataset with skeleton
x_teste_esq = []
count = 0

for image in x_teste_not:
  img_skel = image
  _, img_skel = cv2.threshold(img_skel, 127, 255, cv2.THRESH_OTSU)
  skeleton_new = zhangSuen(img_skel)
  x_teste_esq.append(skeleton_new)
  print(count)
  count += 1
```

In [ ]:

```
# Skeleton verification
plt.imshow(x_teste_esq[150])
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f4a4a753780>
```



INSERTING DESCRIPTORS IN THE TEST SET

Applying the images areas

In [ ]:

```
# Creating the file to insert the areas
base_area = []
for img in x_teste_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + \
                                 cv2.THRESH_OTSU)

  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  area = cv2.contourArea(biggest)
  base_area.append(area)
```

In [ ]:

```
# Crossing area information with existing information
final_teste = final_teste.join(pd.DataFrame(base_area)\
                               ).rename(columns={0:'AREA_OBJ'})
```

Applying image height

In [ ]:

```
# Creating the file to insert the height
base_altura = []
for alt in x_teste_esq:
  x, y, w, h = cv2.boundingRect(alt)
  altura = h
  base_altura.append(altura)
```

In [ ]:

```
# Crossing height information with existing information
final_teste = final_teste.join(pd.DataFrame(base_altura)\
                               ).rename(columns={0:'ALTURA'})
```

Applying the width of the images

In [ ]:

```
# Creating the file to insert the width
base_largura = []
for lar in x_teste_esq:
  x, y, w, h = cv2.boundingRect(lar)
  largura = w
  base_largura.append(largura)
```

In [ ]:

```
# Crossing width information with existing information
final_teste = final_teste.join(pd.DataFrame(base_largura)\
                               ).rename(columns={0:'LARGURA'})
```

Applying the contour area

In [ ]:

```
# Creating the file to insert the contour area
base_area2 = []
for area in x_teste_esq:
  x, y, w, h = cv2.boundingRect(area)
  objeto = w * h
  base_area2.append(objeto)
```

In [ ]:

```
# Crossing contour information with existing information
final_teste = final_teste.join(pd.DataFrame(base_area2)\
                               ).rename(columns={0:'AREA_CONTORNO'})
```

Applying horizontal balancing

In [ ]:

```
# Creating the file to insert the horizontal balancing
balan_hor = []
for hor in x_teste_esq:
  x, y, w, h = cv2.boundingRect(hor)
  horizontal = (64-(w/2))
  balan_hor.append(horizontal)
```

In [ ]:

```
# Crossing horizontal balancing information with existing information
final_teste = final_teste.join(pd.DataFrame(balan_hor)\
                               ).rename(columns={0:'BALAN_HOR'})
```

Applying the vertical balancing

In [ ]:

```
# Creating the file to insert the vertical balancing
balan_ver = []
for ver in x_teste_esq:
  x, y, w, h = cv2.boundingRect(ver)
  vertical = (64-(h/2))
  balan_ver.append(vertical)
```

In [ ]:

```
# Crossing vertical balancing information with existing information
final_teste = final_teste.join(pd.DataFrame(balan_ver)\
                               ).rename(columns={0:'BALAN_VER'})
```

Applying the images ratio

In [ ]:

```
# Creating the file to insert the ratio
base_razao = []
for raz in x_teste_esq:
  x, y, w, h = cv2.boundingRect(raz)
  razao = h/w
  base_razao.append(razao)
```

In [ ]:

```
# Crossing ratio information with existing information
final_teste = final_teste.join(pd.DataFrame(base_razao)\
                               ).rename(columns={0:'RAZAO'})
```

Applying compactness

In [ ]:

```
# Creating the file to insert the compactness
base_compacidade = []

for img in x_teste_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + \
                                 cv2.THRESH_OTSU)

  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  area = cv2.contourArea(biggest)
  perimetro = cv2.arcLength(biggest, True)
  compacidade = ((perimetro**2)/area) if area != 0 else 0
  base_compacidade.append(compacidade)
```

In [ ]:

```
# Crossing compactness information with existing information
final_teste = final_teste.join(pd.DataFrame(base_compacidade)\
                               ).rename(columns={0:'COMPACIDADE'})
```

Applying the contour perimeter

In [ ]:

```
# Creating the file to insert the contour perimeter
base_perimetro = []

for img in x_teste_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + \
                                 cv2.THRESH_OTSU)

  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  perimetro = cv2.arcLength(biggest, True)
  base_perimetro.append(perimetro)
```

In [ ]:

```
# Crossing contour perimeter information with existing information
final_teste = final_teste.join(pd.DataFrame(base_perimetro)\
                               ).rename(columns={0:'PERIM_CONTORNO'})
```

Applying rectangle

In [ ]:

```
# As we already have the area, width and height values calculated in the table,
# we will do the calculation directly
final_teste['RETANG.'] = final_teste['AREA_OBJ'] / (final_teste['ALTURA'] * \
                                                    final_teste['LARGURA'])
```

Applying the Euler number

In [ ]:

```
# Creating the file to insert the Euler number
base_euler = []

for img in x_teste_esq:
  contornos,_ = cv2.findContours(img,cv2.RETR_CCOMP,cv2.CHAIN_APPROX_SIMPLE)

  # We take 2 out of the outline because 1 represents the letter itself and
  # the other 1 is from the formula.
  euler = 2 - len(contornos)
  base_euler.append(euler)
```

In [ ]:

```
# Crossing Euler number information with existing information
final_teste = final_teste.join(pd.DataFrame(base_euler)\
                               ).rename(columns={0:'EULER'})
```

Applying the area and perimeter of the convex closure

In [ ]:

```
# Creating the file to insert the areas
base_area3 = []
base_perimetro2 = []

for img in x_teste_esq:
  _, binary_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + \
                                 cv2.THRESH_OTSU)

  cnt, hierarchy = cv2.findContours(binary_otsu, 1, 2)
  biggest = max(cnt, key = cv2.contourArea)
  hull = cv2.convexHull(biggest)
  area = cv2.contourArea(hull)
  base_area3.append(area)
  perimetro = cv2.arcLength(hull, True)
  base_perimetro2.append(perimetro)
```

In [ ]:

```
# Crossing the area and perimeter information of the closure with the
# existing ones
final_teste = final_teste.join(pd.DataFrame(base_area3)\
                               ).rename(columns={0:'AREA_FECHO'})

final_teste = final_teste.join(pd.DataFrame(base_perimetro2)\
                               ).rename(columns={0:'PERIM_FECHO'})
```

Applying convexity

In [ ]:

```
# As we already have the closing and contour perimeter values in the table,
# we will do the calculation directly
final_teste['CONVEXIDADE'] = final_teste['PERIM_FECHO'] / final_teste['PERIM_CONTORNO']
```

Applying solidity

In [ ]:

```
# As we already have the closure and contour area values in the table,
# we will do the calculation directly
final_teste['SOLIDEZ'] = final_teste['AREA_CONTORNO'] / final_teste['AREA_FECHO']
```

In [ ]:

```
# Validation
final_teste.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img013-00999	img013	262.5	68	95	6460	16.5
1	img013-00985	img013	19.5	88	71	6248	28.5
2	img013-00983	img013	25.5	88	71	6248	28.5
3	img011-00856	img011	978.0	64	86	5504	21.0
4	img012-00847	img012	4432.5	89	82	7298	23.0

Normalizing the dataset

In [ ]:

```
# Scaling the test set
final_teste_scaled = pd.DataFrame(scaler.transform(final_teste.iloc[:,3:17]))
final_teste_scaled.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	
0	-2.476937	1.672933	0.191017	-1.672933	2.476937	-1.462436	-0.263656	0.978087	-0.8
1	0.171420	-0.068661	0.021622	0.068661	-0.171420	-0.079860	0.628012	0.706947	-0.9
2	0.171420	-0.068661	0.021622	0.068661	-0.171420	-0.079860	0.463931	0.925569	-0.9
3	-3.006609	1.019835	-0.572860	-1.019835	3.006609	-1.387461	-0.329020	0.033893	-0.3
4	0.303837	0.729570	0.860608	-0.729570	-0.303837	-0.486650	-0.341456	-0.748966	1.2

Applying K-means

In [ ]:

```
# Predict and already aggregating in the final file
final_teste['pred_y_kmeans'] = kmeans.predict(final_teste_scaled)
```

In [ ]:

```
# Validation
final_teste.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img013-00999	img013	262.5	68	95	6460	16.5
1	img013-00985	img013	19.5	88	71	6248	28.5
2	img013-00983	img013	25.5	88	71	6248	28.5
3	img011-00856	img011	978.0	64	86	5504	21.0
4	img012-00847	img012	4432.5	89	82	7298	23.0

Applying Random Forest

In [ ]:

```
# Predict and already aggregating in the final file
final_teste['pred_y_rf'] = model_rf.predict(final_teste_scaled)
```

In [ ]:

```
# Validation
final_teste.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img013-00999	img013	262.5	68	95	6460	16.5
1	img013-00985	img013	19.5	88	71	6248	28.5
2	img013-00983	img013	25.5	88	71	6248	28.5
3	img011-00856	img011	978.0	64	86	5504	21.0
4	img012-00847	img012	4432.5	89	82	7298	23.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by Kmeans
pd.crosstab(final_teste["ANOTACAO"],final_teste["pred_y_kmeans"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_kmeans	0	1	2
ANOTACAO			
img011	68.780488	14.146341	17.073171
img012	54.634146	7.317073	38.048780
img013	64.390244	5.365854	30.243902

In [ ]:

```
# Cross-validation between the label and the Random Forest
pd.crosstab(final_teste["ANOTACAO"],final_teste["pred_y_rf"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_rf	A	B	C
ANOTACAO			
img011	96.097561	1.951220	1.951220
img012	3.414634	94.146341	2.439024
img013	3.414634	1.951220	94.634146

Creating the HOG file (test) and applying K-means and random forest in it

In [ ]:

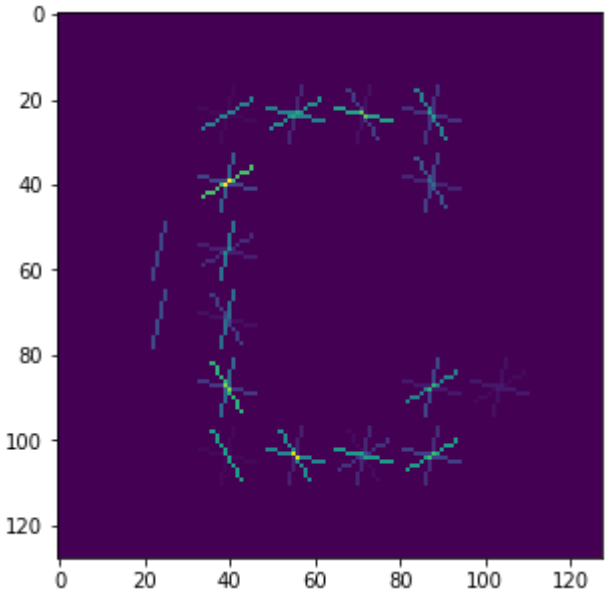
```
# HOG
base_hog = []

for img in x_teste_esq:
  fd, hog_image = hog(img, orientations=8, pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1), visualize=True, multichannel=False)
  base_hog.append(hog_image)
```

In [ ]:

```
# HOG validation
plt.imshow(base_hog[150])
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f4a4fa431d0>
```



In [ ]:

```
# Adjusting the dimensions of the HOG file for the application of k-means
base_hog_ok = np.array(base_hog)
nsamples, nx, ny = base_hog_ok.shape
base_hog_final = base_hog_ok.reshape((nsamples,nx*ny))
```

Applying K-means HOG

In [ ]:

```
# K-means
final_teste['pred_y_kmeans_hog'] = kmeans_hog.predict(base_hog_final)
```

In [ ]:

```
# Validation
final_teste.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img013-00999	img013	262.5	68	95	6460	16.5
1	img013-00985	img013	19.5	88	71	6248	28.5
2	img013-00983	img013	25.5	88	71	6248	28.5
3	img011-00856	img011	978.0	64	86	5504	21.0
4	img012-00847	img012	4432.5	89	82	7298	23.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by Kmeans_hog
pd.crosstab(final_teste["ANOTACAO"],final_teste["pred_y_kmeans_hog"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_kmeans_hog	0	1	2
ANOTACAO			
img011	99.512195	0.487805	0.000000
img012	16.585366	2.439024	80.975610
img013	14.146341	85.365854	0.487805

Applying Random Forest HOG

In [ ]:

```
# Predict in the HOG file and already adding the column at the end
final_teste['pred_y_rf_hog'] = model_rf_hog.predict(base_hog_final)
```

In [ ]:

```
# Validation
final_teste.head()
```

Out[ ]:

	IMAGEM	ANOTACAO	AREA_OBJ	ALTURA	LARGURA	AREA_CONTORNO	BALAN_HOR
0	img013-00999	img013	262.5	68	95	6460	16.5
1	img013-00985	img013	19.5	88	71	6248	28.5
2	img013-00983	img013	25.5	88	71	6248	28.5
3	img011-00856	img011	978.0	64	86	5504	21.0
4	img012-00847	img012	4432.5	89	82	7298	23.0

In [ ]:

```
# Cross-validation between the label and the groups discovered by RF_hog
pd.crosstab(final_teste["ANOTACAO"],final_teste["pred_y_rf_hog"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_rf_hog	A	B	C
ANOTACAO			
img011	98.536585	1.463415	0.000000
img012	3.902439	95.121951	0.975610
img013	3.902439	0.000000	96.097561

In [ ]:

```
# Downloading the final files
final_treino.to_csv(r'/content/base_final_treino.csv')
final_teste.to_csv(r'/content/base_final_teste.csv')
```

CONCLUSION

At the end of this project, I concluded that the models that use HOG \ performed much better than the models with the descriptors extracted \ separately. This occurred for both models, k-means and Random Forest.

It is worth mentioning that the random forest performed better when \ compared to k-means, both for the version of descriptors extracted \ separately and for the HOG.