Aluno: Matheus Felipe Régis
Instituto Federal Catarinense - Campus Videira
13/08/2020

Detecção de Canecas utilizando Classificadores com OpenCV

1. Ferramentas Utilizadas

Para realizar este projeto, foram utilizados as ferramentas PyCharm, e Cmder, e a linguagem Python. Também utilizada a biblioteca OpenCV, utilizando o HaarCascade, um método de detecção de objetos proposto pelo OpenCV(há vários outros métodos dentro do próprio OpenCV, nesse projeto foi abordado somente o HaarCascade).

2. Funcionamento HaarCascade

O HaarCascade necessita de imagens positivas onde essas imagens são objetos na qual você quer detectar, e imagens negativas, onde são imagens/objetos aleatórios que podem ou não estar relacionado com o objeto, Por exemplo: Poderia ter uma imagem de uma mesa, em que poderia ter uma caneca, para classificá-la como imagens negativas a caneca não pode estar nessa mesa.
O HaarCascade utiliza o algoritmo de AdaBoost para o treinamento do classificador. O treinamento do algoritmo é a seleção do melhor conjunto de características da imagem, utilizando as características abaixo. Esses quadrados/retangulos, cada metade representa um pixel, podendo então ter características que irão filtrar de dois pixeis, três pixeis, ou quatro pixeis ao mesmo tempo, o controle e a melhor solução para a identificação, e detecção da imagem. é feito pelo algoritmo AdaBoost.

Figura 1 - Seleção das Características de uma imagem com o algoritmo AdaBoost.

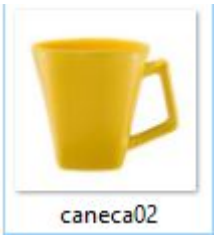
O resultado do treinamento do Classificador será um arquivo .xml, onde contém as informações propostas de cada característica propostas para a melhor solução.

3. Treinamento do Classificador

Nessa etapa será utilizado o Cmder, que é um console que emula o linux.

```
opencv_createsamples -img caneca02.jpg -bg negativas/bg.txt -info positivas02/positivas02.lst -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 - w 48 -h 48 -num 300 -bgcolor 255 -bgthresh 8
```

O primeiro comando consiste em criar um banco de imagens positivas, através das imagens negativas…
-img caneca02.jpg consiste no caminho/arquivo em que será utilizado para criar as imagens positivas,
-bg negativas/bg.txt é o caminho onde está o arquivo, listando as mais de 3000 imagens negativas dentro da pasta negativas,
-info positivas02 que indica a pasta onde irá ser criado as imagens positivas, e as informações das imagens será geradas dentro do arquivo positivas02.lst, o
-maxxangle 0.5, -maxyangle 0.5 e -maxzangle 0.5 servem para girar a imagem da caneca positiva em vários ângulos, conforme as figuras abaixo:



caneca original,  imagem da positiva da caneca

Os parâmetros -w 48 e -h 48 são o tamanho da imagem positiva(a caneca) jogadas dentro do tamanho das imagens negativas(a foto com o fundo que tem 100x100)
-num serve para dizer que serão criadas 300 imagens positivas
-bgcolor 255 e -bgthresh 8 tem como objetivo tirar o fundo branco da imagem da caneca original para não dar alteração nas imagens positivas.

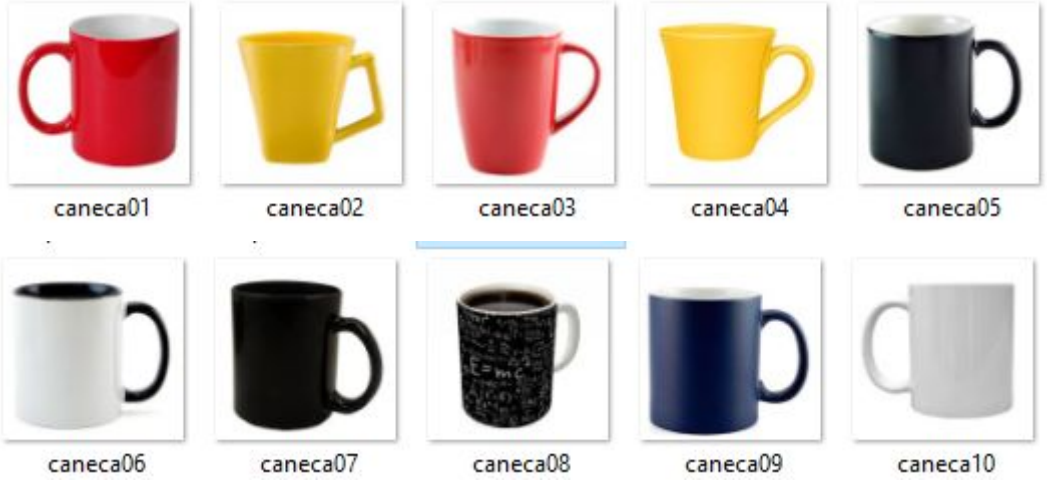
Após executado o código, será gerado as fotos e o arquivo positiva02.lst que será utilizado para o próximo comando do treinamento do classificador.

```
opencv_createsamples -info positivas02/positivas02.lst -num 2000 -w 20 -h 20 -vec vetor02.vec
```

Esse comando tem como finalidade a criação de um vetor que será utilizado para o treinamento
`-info positivas02/positivas02.lst` serve para acessar a pasta info e o arquivo onde contém todas as informações das imagens positivas
`-num 2000` número de imagens

`-w 20 -h 20` dimensão das imagens(quanto maior mais demora o treinamento do algoritmo)
`-vec vetor02.vec` o nome do arquivo vetor gerado que será gerado.

Como foram utilizados 10 imagens base para a criação das imagens positivas, sendo as imagens bases:



foram rodados os comandos acima 10x, assim criando um banco de 3000 imagens positivas e gerados 10 arquivos vec.
Como o HaarCascade através do próximo comando opencv_traincascade, lê somente um vetor, foi necessário a utilização de um arquivo chamado mergevec.py que faz a junção de todos os vetores das imagens positivas, através do comando:
`python mergevec.py -v vec/ -o vetor_final.vec`

opencv_traincascade -data classificador -vec vetor_final.vec -bg bg.txt -numPos 1800 -numNeg 1200 -numStages 10 -w 20 -h 20 -precalcValBufSize 1024 -precalcIdxBufSize 1024

Por fim o comando de treino haarcascade.
-data classificador será o arquivo classificador.xml já treinado.

-vec vetor_final.vec busca o arquivo vetor_final onde é o arquivo de junção dos 10 vetores de imagens positivas.

-bg bg.txt o arquivo de imagens negativas dentro da pasta onde contem todas as imagens negativas.

-numPos 1800 número de imagens positivas

-numNeg 1200 número de imagens negativas

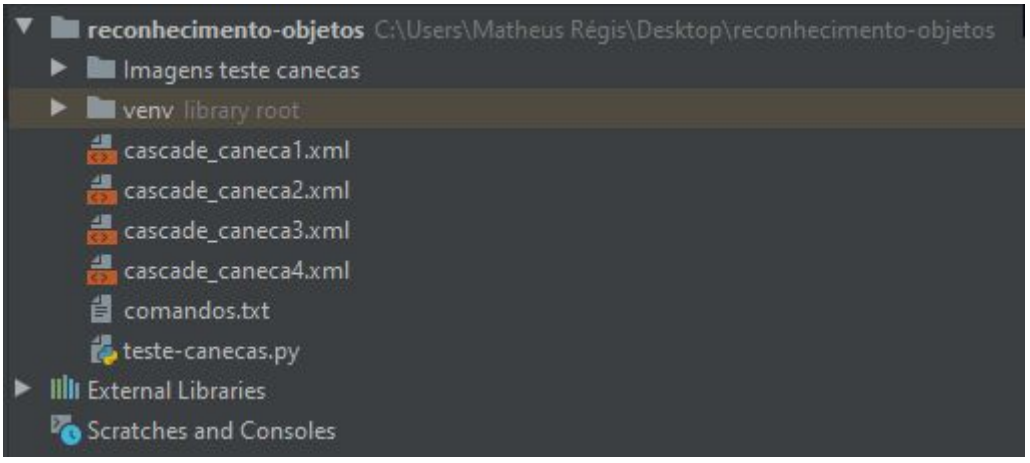
-numStages 10 número de estágios do treino do classificador

-w 20 -h 20 tamanho da imagem que está sendo treinada (deve ser o mesmo valor utilizadas em cada imagem positiva na criação do vetor)

-precalcValBufSize 1024 -precalcIdxBufSize 1024 buffer de memória tanto GPU quanto memória ram para o treinamento, quanto maior o valor menor o tempo de treinamento do classificador.

4. Pastas e Arquivos do Projeto

O projeto foi separado conforme a imagem abaixo:



Sendo o arquivo onde contém o algoritmo, chamado teste-canecas.py os arquivos cascade_caneca*.xml são os arquivos treinados durante esse projeto, o

comandos.txt é os comandos do cmder e a pasta imagens teste são as imagens que serão feito os testes dos classificadores, sendo 10 imagens de testes.

5. Implementação do Algoritmo

Para iniciar a implementação é necessário a importação da biblioteca OpenCV, através do comando: `import cv2`
Essa biblioteca será usada para chamada de outras funções que compõe a biblioteca OpenCV durante toda a implementação do algoritmo.

```
imagem1 = cv2.imread('Imagens teste canecas/teste02.jpg')
```
Após realizar a importação, deve-se criar uma variável em que fará a leitura da imagem, passando a localização de seu diretório.

```
classificador1 = cv2.CascadeClassifier('cascade_caneca1.xml')
```
a biblioteca está chamando a função CascadeClassifier que basicamente carrega os dados treinados pelo algoritmo AdaBoost conforme explicado anteriormente. Esses dados estão sendo armazenados na variável classificador1.

```
imagemCinza1 = cv2.cvtColor(imagem1, cv2.COLOR_BGR2GRAY)
```
Nessa parte, a variável imagem1 está sendo transformada na coloração GRAY e armazenada na variável imagemCinza1.

```
deteccoes1 = classificador1.detectMultiScale(imagemCinza1,
scaleFactor=1.2, minNeighbors=4)
```
Nessa parte do algoritmo o classificador1 está chamando a função detectMultiScale, também do OpenCV, que tem como finalidade detectar possíveis canecas dentro da imagemCinza1. Lembrando que essa detecção deve ser filtrada, podendo inclusive passar como parâmetros alguns fatores que ajudarão e irão melhorar a detecção do objeto.
Os parâmetros scaleFactor=1.2, tem como finalidade aumentar/diminuir o fator de escala da imagem, ou seja, caso uma caneca esteja em frente da camera, e possua outra caneca mais ao fundo, a imagem em frente a camera possuirá uma escala muito maior que a imagem ao fundo, e talvez o algoritmo não consiga reconhecer que ao fundo, também possui uma caneca, esse parâmetro tem como finalidade deixar a ambas as canecas com tamanho parecido, aumentando/diminuindo a escala da imagem.
O segundo parâmetro, determina quantos BoundingBox vizinhos o objeto vai ter na vizinhança, por exemplo, a imagem possui uma caneca, porém o algoritmo detectou 5, caso eu determine que o minNeighbors dessa imagem é 5, ele irá detectar na

vizinhança, o mínimo de 5 vizinhos, qual o melhor BoundingBox se encaixa para a detecção do objeto. Quanto maior o valor do minNeighbors, melhor a qualidade da detecção.

```
print(deteccoes1)
```

Esse comando está printando no console, a variavel deteccoes, ou seja, ela vai mostrar as detecções conforme a figura abaixo:

Para a seguinte imagem:

```
[[ 37  86  34  34]
 [178  67  31  31]
 [ 38  51  21  21]
 [170  94  26  26]
 [ 54 265  37  37]
 [202 237  52  52]
 [192 123  41  41]
 [217 317  30  30]
 [293 141  37  37]]
```

O algoritmo detectou 9 BoundingBox. Os valores apresentados dentro da matriz da figura acima correspondem nas linhas as seguintes finalidades, X, do plano cartesiano de uma imagem, Y, também do plano cartesiano de uma imagem, L que é a largura e A de altura, com isso, conseguimos extrair que o primeiro

boundingBox, onde no X = 37 pixeis, Y = 86 pixeis, L=34pixeis e A = 34Pixeis,

corresponde a seguinte imagem:
Logo esse BoundingBox é um falso positivo, onde ele não detecta corretamente a caneca

```
for (x, y, l, a) in deteccoes1:
    cv2.rectangle(imagem1, (x, y), (x + l, y + a), (0, 255, 0), 2)
```

Como foram detectados 9 BoundingBox, o for serve para percorrer todos os valores e desenhar através do comando cv2.rectangle na imagem1 , a partir do X a largura de l, e a partir de y, a largura de a, uma caixa(BoundingBox) que possua a cor verde.

```
cv2.imshow('Classificador 1', imagem1)
```

por fim, o comando imshow mostra a imagem já com os BoudingBox colocados dentro da imagem.

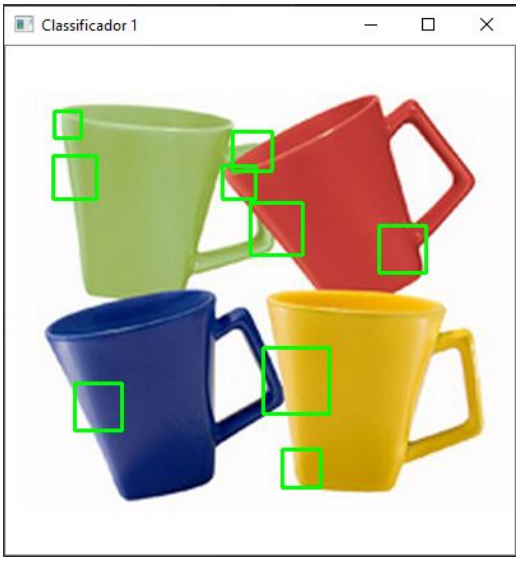
```
cv2.waitKey(0)
```

Esse comando faz com que para fechar a janela que abre a imagem do comando com a função imshow, eu necessite apertar qualquer tecla do teclado.

```
cv2.destroyAllWindows()
```

E por fim deletar tudo salvo na memória/imagem.

O Resultado fica esse:



Como podemos ver, o classificador não acertou nenhuma detecção correta da caneca, possuindo vários falsos negativos, isso ocorre pois está faltando mais imagens negativas para a detecção desse objeto.

6. Conclusão

Para a conclusão deste projeto, foram realizados quatro treinos de classificadores para detecção de canecas, onde em cada treino foram utilizados parâmetros diferentes, altura e largura da imagem positiva, quantidade de imagens positivas, quantidade de imagens negativas, tamanho do buffer da memória, conforme especificado abaixo:

Classificador 1:

Para o classificador 1 foi utilizado somente a caneca01 de base e sendo assim só criado um vetor com os seguintes parâmetros:

```
opencv_createsamples -img caneca01.png -bg negativas/bg.txt -info
positivas/positivas.lst -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5-num 1800
-bgcolor 255
```

```
opencv_createsamples -info positivas/positivas.lst -num 1800 -w 18 -h 18 -vec
positivas.vec
```

```
opencv_traincascade -data classificador -vec positivas.vec -bg bg.txt -numPos 1800
-numNeg 800 -numStages 10 -w 18 -h 18 -precalcValBufSize 1024
-precalcIdxBufSize 1024
```

Para o treinamento desse classificador levaram cerca de 1 minuto

Classificador 2:

Para o classificador 2 foi utilizado as 10 imagens de canecas de base:

```
opencv_createsamples -img caneca01.png -bg negativas/bg.txt -info
positivas01/positivas01.lst -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -w 48 -h
48 -num 1800 -bgcolor 255 -bgthresh 8
```

Repetindo 10x esse mesmo código e gerando 10 pastas com imagens positivas

```
opencv_createsamples -info positivas01/positivas01.lst -num 2000 -w 20 -h 20 -vec positivas01.vec
```
nesse caso foram utilizados 2000 fotos com tamanhos 20x20

```
python mergevec.py -v vec/ -o vetor_final.vec
```

```
opencv_traincascade -data classificador -vec positivas.vec -bg bg.txt -numPos 1800 -numNeg 1200 -numStages 10 -w 20 -h 20 -precalcValBufSize 1024 -precalcIdxBufSize 1024
```

Para esse treinamento, foram utilizados mais imagens negativas e o tamanho das imagens positivas de 20x20.

Para o treinamento desse classificador levaram cerca de 7 minutos

Classificador 3:

A partir do classificador 3 foram utilizados as mesmas imagens positivas, porém mudados os valores somente dos vetores:

```
opencv_createsamples -info positivas01/positivas01.lst -num 2000 -w 24 -h 24 -vec positivas01.vec
```
nesse caso foram utilizados 2000 fotos com tamanhos 24x24, parece uma diferença de somente 4x4 pixeis, porém isso aumentou consideravelmente o tempo de treinamento

```
python mergevec.py -v vec/ -o vetor_final.vec
```

```
opencv_traincascade -data classificador -vec positivas.vec -bg bg.txt -numPos 2000 -numNeg 3200 -numStages 15 -w 24 -h 24 -precalcValBufSize 8192 -precalcIdxBufSize 8192
```

Para esse treinamento, foram utilizados mais imagens negativas e o tamanho das imagens positivas de 24x24.

Para o treinamento desse classificador levaram cerca de 1 hora e 11 minutos.

Classificador 4:

```
opencv_createsamples -info positivas01/positivas01.lst -num 1800 -w 24 -h 24 -vec positivas01.vec
```

```
python mergevec.py -v vec/ -o vetor_final.vec
```

```
opencv_traincascade -data classificador -vec positivas.vec -bg bg.txt -numPos 1800 -numNeg 1200 -numStages 15 -w 24 -h 24 -precalcValBufSize 8192 -precalcIdxBufSize 8192
```

Para esse treinamento, foram utilizados mais imagens negativas e o tamanho das imagens positivas de 24x24.

Para o treinamento desse classificador levaram cerca de 45 minutos.

Por fim, a imagem que mostra o resultado dos 4 classificadores: