

Visão Computacional

Jorge Marques Ferreira Junior

12 de janeiro de 2024

Conteúdo

| | | |
|----------|--|-----------|
| 1 | Antes de começar - alguns termos importantes | 2 |
| 2 | Detecção de Objetos em imagem - YOLO e darknet | 3 |
| 2.1 | Teste Simples | 3 |
| 2.2 | Criação do Dataset | 6 |
| 2.3 | Treinamento da Rede Neural - Google Colab. | 8 |
| 2.4 | Detecção | 12 |
| 3 | Detecção de Objetos em imagen - YOLO, darknet e OpenCV | 12 |
| 3.1 | Detecção Simples | 12 |
| 3.2 | Contagem de Objetos | 16 |
| 4 | Detecção de Objetos em vídeo - YOLO e darknet | 23 |
| 5 | Detecção de Objetos em vídeo - YOLO, darknet e OpenCV | 24 |
| 6 | Guia - Fundamentos OpenCV | 24 |
| 6.1 | Introdução | 24 |
| 6.1.1 | <i>imread()</i> , <i>imshow()</i> , <i>waitKey()</i> | 25 |
| 6.1.2 | <i>VideoCapture()</i> | 26 |
| 6.1.3 | <i>set()</i> | 26 |
| 6.1.4 | <i>flip()</i> | 27 |
| 6.2 | Funções Básicas | 27 |
| 6.2.1 | Convertendo uma imagem para a escala de cinza | 27 |
| 6.2.2 | Filtro Gaussiano, eliminando ruídos de imagens | 28 |
| 6.2.3 | Filtro Canny, detecção de bordas | 28 |
| 6.2.4 | Filtro para aumentar o realce de bordas - dilate | 30 |
| 6.2.5 | Filtro para diminuir o realce de bordas - erode | 31 |
| 6.3 | Redimensionando e Recortando | 32 |
| 6.4 | Formas e Textos | 34 |
| 6.5 | Distorções e Perspectivas | 36 |
| 6.6 | Juntando Imagens | 37 |
| 6.7 | Detecção de Cores | 38 |
| 6.8 | Contornos/Detecção de Formas | 39 |

1 Antes de começar - alguns termos importantes

Antes de iniciarmos o estudo de detecções de objetos vamos estudar alguns conceitos que estarão presentes em todo momento e que devemos compreender com destreza para que nosso trabalho seja bem sucedido.

- **Threshold:** O limiar (threshold) no contexto do YOLO refere-se a um valor crítico que determina quão confiante o modelo deve estar em uma detecção para considerá-la válida.

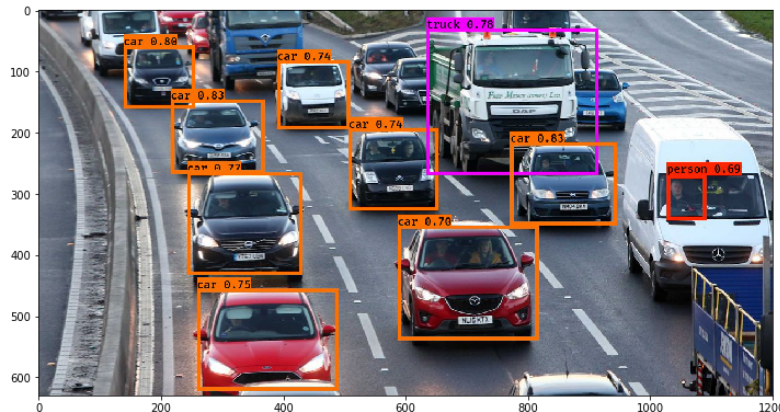


Figura 1: Threshold mínimo de 69%.

Aplicação no YOLO: Durante a fase de inferência, onde o modelo faz previsões em novas imagens, as detecções com uma pontuação de confiança (probabilidade) inferior ao limiar são descartadas. Isso ajuda a filtrar detecções menos confiáveis, contribuindo para resultados mais precisos.

- **Non-Maxima Suppression:** A supressão de não-máximo é um procedimento pós-processamento usado para eliminar detecções redundantes e manter apenas as detecções mais confiantes.

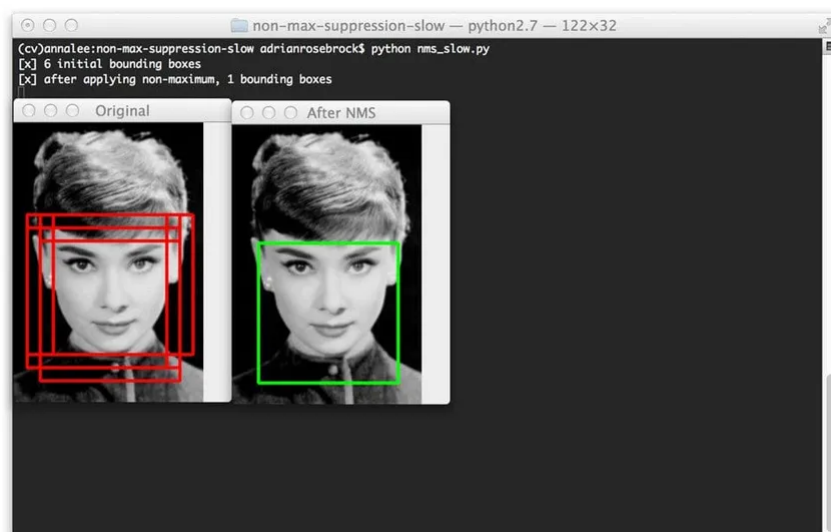
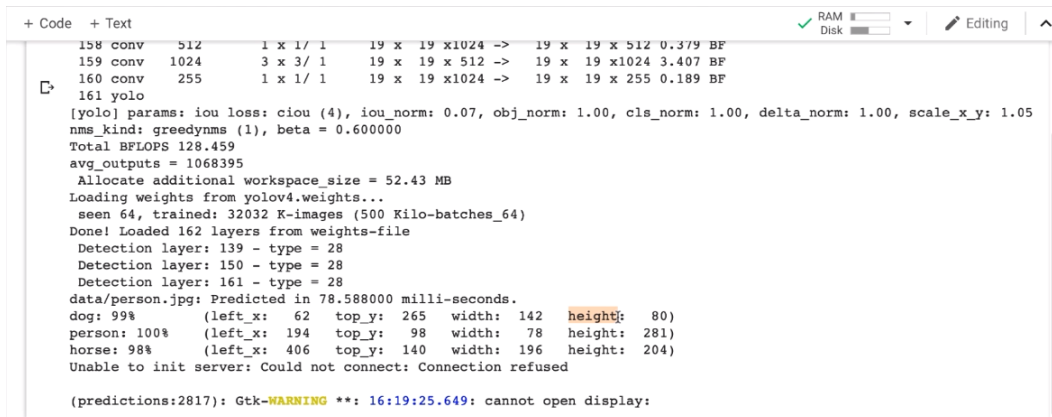


Figura 2: Supressão de não-máximo na prática.

Aplicação no YOLO: Como o YOLO pode gerar múltiplas caixas delimitadoras para um objeto em uma única detecção, a supressão de não-máximo ajuda a remover caixas sobrepostas ou duplicadas. O algoritmo classifica as detecções com base na confiança e, em seguida, iterativamente elimina as caixas que têm uma sobreposição significativa com detecções mais confiantes. O resultado final é uma lista de detecções não redundantes.

- **ext_output:** Refere-se a uma saída estendida ou informações adicionais geradas durante o processo de detecção de objetos pelo YOLO. Isso pode incluir detalhes sobre as caixas delimitadoras, pontuações de confiança, classes previstas ou outros parâmetros relacionados à detecção de objetos..



```
+ Code + Text
158 conv 512 1 x 1/ 1 19 x 19 x1024 -> 19 x 19 x 512 0.379 BF
159 conv 1024 3 x 3/ 1 19 x 19 x 512 -> 19 x 19 x1024 3.407 BF
160 conv 255 1 x 1/ 1 19 x 19 x1024 -> 19 x 19 x 255 0.189 BF
161 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedy (1), beta = 0.600000
Total BFLOPS 128.459
avg_outputs = 1068395
Allocate additional workspace_size = 52.43 MB
Loading weights from yolov4.weights...
seen 64, trained: 32032 K-images (500 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
data/person.jpg: Predicted in 78.588000 milli-seconds.
dog: 99% (left_x: 62 top_y: 265 width: 142 height: 80)
person: 100% (left_x: 194 top_y: 98 width: 78 height: 281)
horse: 98% (left_x: 406 top_y: 140 width: 196 height: 204)
Unable to init server: Could not connect: Connection refused

(predictions:2817): Gtk-WARNING **: 16:19:25.649: cannot open display:
```

Figura 3: Visualizando o tamanho das caixas delimitadoras.

Aplicação no YOLO:.

2 Detecção de Objetos em imagem - YOLO e darknet

Para treinarmos uma rede neural a fim de detectar um determinado objeto, precisamos primeiramente criar um dataset de imagens para treino e teste. Se você não tiver as imagens, existem alguns sites que facilitam o processo para a criação desse dataset. Nesse caso utilizaremos o site **Open Images Dataset V7**.

Esse site contém inúmeras imagens de varias classes e para baixar tais imagens utilizaremos uma ferramenta mantida pelo Google que facilita esse processo. Essa ferramenta pode ser encontrada através do **GitHub-OIDv4Toolkit**.

2.1 Teste Simples

Existe uma base de dados já treinada que aborda 80 classes diferentes de objetos, os tipos dessas classes podem ser encontrados no site **Database-Coco**. Dependendo da precisão que você busca para a predição de imagens e se caso o objeto específico a ser detectado estiver dentre as classes de objetos da base de dados coco, então podemos testar a predição da imagem sem a necessidade de treinarmos uma rede neural antecipadamente.

Vamos fazer uma detecção simples a partir de um **modelo pré-treinado**. Link de acesso ao código fonte: **DeteccaoSimples.ipynb**.

```

1 #Deteccao-Simples
2
3 import tensorflow as tf
4 device = tf.test.gpu_device_name()
5
6 if device:
7     print(f"GPU encontrada: {device}")
8 else:
9     print("Nenhuma GPU disponivel.")
10
11
12 ##Etapa1 - Download da darknet
13
14 !git clone https://github.com/AlexeyAB/darknet.git
15
16 ##Etapa2 - Compilando a biblioteca
17
18 cd /content/darknet
19
20 # essa linha de codigo garante que a darknet possa ser recompilada
    caso necessario
21 # uma vez compilada a darknet , nao basta reiniciar a celula de
    execucao do colab para refazer o processo,
22 # eh preciso rodar essa linha de codigo abaixo:
23 rm -rf ./obj/image_opencv.o ./obj/http_stream.o ./obj/gemm.o ./obj/
    utils.o ./obj/dark_cuda.o ./obj/convolutional_layer.o ./obj/list
    .o ./obj/image.o ./obj/activations.o ./obj/im2col.o ./obj/col2im
    .o ./obj/blas.o ./obj/crop_layer.o ./obj/dropout_layer.o ./obj/
    maxpool_layer.o ./obj/softmax_layer.o ./obj/data.o ./obj/matrix.
    o ./obj/network.o ./obj/connected_layer.o ./obj/cost_layer.o ./
    obj/parser.o ./obj/option_list.o ./obj/darknet.o ./obj/
    detection_layer.o ./obj/captcha.o ./obj/route_layer.o ./obj/
    writing.o ./obj/box.o ./obj/nightmare.o ./obj/
    normalization_layer.o ./obj/avgpool_layer.o ./obj/coco.o ./obj/
    dice.o ./obj/yolo.o ./obj/detector.o ./obj/layer.o ./obj/compare
    .o ./obj/classifier.o ./obj/local_layer.o ./obj/swag.o ./obj/
    shortcut_layer.o ./obj/activation_layer.o ./obj/rnn_layer.o ./
    obj/gru_layer.o ./obj/rnn.o ./obj/rnn_vid.o ./obj/crnn_layer.o
    ./obj/demo.o ./obj/tag.o ./obj/cifar.o ./obj/go.o ./obj/
    batchnorm_layer.o ./obj/art.o ./obj/region_layer.o ./obj/
    reorg_layer.o ./obj/reorg_old_layer.o ./obj/super.o ./obj/voxel.
    o ./obj/tree.o ./obj/yolo_layer.o ./obj/gaussian_yolo_layer.o ./
    obj/upsample_layer.o ./obj/lstm_layer.o ./obj/conv_lstm_layer.o
    ./obj/scale_channels_layer.o ./obj/sam_layer.o darknet
24
25 # ativando a GPU e o OPENCV como suporte a darknet
26 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
27 !sed -i 's/GPU=0/GPU=1/' Makefile
28 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
29
30 !make
31
32 ##Etapa3 - Baixando os pesos pre-treinados
33
34 !wget https://github.com/AlexeyAB/darknet/releases/download/
    darknet_yolo_v3_optimal/yolov4.weights
35
36 ##Etapa4 - Testando o detector
37
38 !./darknet detect cfg/yolov4.cfg yolov4.weights data/person.jpg

```

```

39
40 ##Etapa5 - Visualizando a imagem
41
42 import cv2
43 import matplotlib.pyplot as plt
44
45 def mostrar(caminho):
46     imagem = cv2.imread(caminho)
47     fig = plt.gcf()
48     fig.set_size_inches(18,10)
49     plt.axis = ("off")
50     plt.imshow(cv2.cvtColor(imagem, cv2.COLOR_BGR2RGB))
51     plt.show()
52
53
54 mostrar('/content/darknet/predictions.jpg')
55
56 ##Etapa6 - Testando com qualquer imagem...
57
58 !./darknet detect cfg/yolov4.cfg yolov4.weights /content/darknet/
    data/eagle.jpg
59
60 mostrar('/content/darknet/predictions.jpg')
61
62 ##Etapa7 - Facilitando nosso trabalho - Algumas funcoes
    interessantes
63
64 import os
65 def detectar(imagem):
66     os.system(f"cd /content/darknet && ./darknet detect cfg/yolov4.cfg
        yolov4.weights {imagem}")
67     mostrar('/content/darknet/predictions.jpg')
68
69
70 import os
71 def listar_jpg(caminho):
72     return [os.path.join(caminho, file) for file in os.listdir(caminho)
        if file.lower().endswith('.jpg')]
73
74 detectar('/content/darknet/data/dog.jpg')
75
76
77
78 for image in listar_jpg('/content/darknet/data'):
79     detectar(image)
80
81
82 ##Etapa8 - Carregando fotos personalizadas do drive
83
84 from google.colab import drive
85 drive.mount('/content/gdrive')
86
87 detectar('/content/gdrive/MyDrive/ColabNotebooks/CatDetector/tito3.
    jpg')
88
89 ##Etapa9 - Definindo um percentual minimo aceito para deteccao (
    threshold):
90
91 !./darknet detect cfg/yolov4.cfg yolov4.weights /content/darknet/
    data/horses.jpg -thresh 0.98

```

```

92
93 mostrar('/content/darknet/predictions.jpg')
94
95 ##Etapa10 - Obtendo as posicoes dos animais detectados (ext_output)
96
97 !./darknet detect cfg/yolov4.cfg yolov4.weights /content/darknet/
    data/horses.jpg -ext_output
98
99 mostrar('/content/darknet/predictions.jpg')
100
101 ##Etapa11 - Salvar no drive os modelos pre-treinados
102
103 !zip -r modelo_YOLOv4.zip yolov4.weights cfg/yolov4.cfg cfg/coco.
    names
104
105 !cp modelo_YOLOv4.zip /content/gdrive/MyDrive/yolov4/modelo_YOLOv4.
    zip
106
107 ## Detectando objeto com outros modelos treinados
108
109 ### Download dos pesos (da internet)
110
111 !wget https://pjreddie.com/media/files/yolov3-openimages.weights
112
113 ls
114
115 ls cfg
116
117 !./darknet detector test cfg/openimages.data cfg/yolov3-openimages.
    cfg yolov3-openimages.weights data/dog.jpg
118
119 mostrar('predictions.jpg')
120
121 !wget https://github.com/AlexeyAB/darknet/releases/download/
    darknet_yolo_v4_pre/yolov4-tiny.weights
122
123 ls
124
125 ls cfg
126
127 !./darknet detector test cfg/coco.data cfg/yolov4-tiny.cfg yolov4-
    tiny.weights data/dog.jpg
128
129 mostrar('predictions.jpg')
130

```

Listing 1: Detecção a partir de um modelo pré-treinando.

2.2 Criação do Dataset

Demonstrando a aplicação do código para download de imagens no ambiente do Google Colab utilizando a ferramenta OIDv4ToolKit. Link para acesso direto ao código fonte: **ImageDownloader.ipynb**.

```

1 #Dataset Creator
2
3 ##01- Ativar GPU e Acessar Google Drive
4
5 import tensorflow as tf
6 GPU = tf.test.gpu_device_name()

```

```

7 print(GPU)
8
9 # Saida esperada: /device:GPU:0
10
11 from google.colab import drive
12 drive.mount('/content/gdrive')
13
14 # Saida esperada: Mounted at, /content/gdrive
15
16 ##02- Baixar ferramenta para download de imagens, bibliotecas
    necessarias e conversor de anotacoes yolo
17
18 !git clone https://github.com/EscVM/OIDv4_ToolKit.git
19
20 cd /content/OIDv4_ToolKit
21
22 !pip3 install -r /content/OIDv4_ToolKit/requirements.txt
23
24 cd /content/
25
26 !git clone https://github.com/jorgemarquesferreirajunior/
    converter_annotations_yolo.git
27
28 !cp /content/converter_annotations_yolo/converter_annotations.py /
    content/OIDv4_ToolKit
29
30 ##03- Baixar imagens desejadas
31
32 cd /content/OIDv4_ToolKit
33
34 !python main.py downloader --classes Pig --type_csv train --limit
    2000 --multiclasses 1 --y
35
36 !python main.py downloader --classes Pig --type_csv test --limit
    400 --multiclasses 1 --y
37
38 ##04- Compactar e salvar imagens no drive
39
40 cd/content/OIDv4_ToolKit/
41
42 !zip -r /content/gdrive/MyDrive/ColabNotebooks/CatDetector/dataset.
    zip /content/OIDv4_ToolKit/OID
43
44 !cp -r /content/OIDv4_ToolKit/OID /content/drive/MyDrive/
    ColabNotebooks/PigDetector
45

```

Listing 2: Baixando dataset de imagens no Colab.

Observações: a quantidade de imagens para teste não deve ultrapassar 20% da quantidade de imagens para treino. Exemplo: 1000 imagens de treino + 200 imagens de teste. Não esqueça também de salvar o dataset em um local seguro como o google drive o então em um dispositivo de armazenamento pois tudo que for carregado apenas no ambiente de execução do google colab será descartado após muito tempo de inatividade na página.

2.3 Treinamento da Rede Neural - Google Colab.

Agora que já temos nosso dataset precisaremos preparar alguns arquivos de configuração da darknet e também renomear algumas pastas para funcionamento adequado de alguns scripts python.

Separamos essa etapa em outro documento no google colab para que ficasse mais fácil o entendimento de cada linha de programação, o link para acesso direto ao código fonte é: **Treinamento.ipynb**.

```
1 #Training Model
2
3 ##01- Ativar GPU e Acessar Google Drive
4
5 import tensorflow as tf
6 GPU = tf.test.gpu_device_name()
7 print(GPU)
8
9 # Saida esperada: /device:GPU:0
10
11 from google.colab import drive
12 drive.mount('/content/gdrive')
13
14 # Saida esperada: Mounted at, /content/gdrive
15
16 ##02- Baixar ferramentas
17
18 !git clone https://github.com/EscVM/OIDv4_ToolKit.git
19
20 cd /content/OIDv4_ToolKit
21
22 !pip3 install -r /content/OIDv4_ToolKit/requirements.txt
23
24 cd /content/
25
26 !git clone https://github.com/jorgemarquesferreirajunior/
    converter_annotations_yolo.git
27
28 !git clone https://github.com/AlexeyAB/darknet
29
30 !cp /content/converter_annotations_yolo/converter_annotations.py /
    content/OIDv4_ToolKit
31
32 ##03- Descompactar dataset do drive para a pasta OIDv4_ToolKit
33
34 !unzip /content/gdrive/MyDrive/ColabNotebooks/CatDetector/dataset.
    zip -d /content/OIDv4_ToolKit/
35
36 !mv /content/OIDv4_ToolKit/content/OIDv4_ToolKit/OID /content/
    OIDv4_ToolKit/OID
37
38 !rm -rf /content/OIDv4_ToolKit/content
39
40 ##04- Configurar arquivo de classes da pasta OIDv4_ToolKit e
    executar o script para converter as anotacoes
41
42 cd /content/OIDv4_ToolKit/
43
44 !cat classes.txt
45
46 !echo -e 'Cat' > classes.txt # Adicionar os nomes das classes
```



```

desejadas
47
48 !python converter_annotations.py
49
50 ##05- Renomear as pastas que contem as imagens de teste e de treino
51
52 # eh necessario renomear as pastas para que a compilacao da darknet
    funcione adequadamente
53 !mv /content/OIDv4_ToolKit/OID/Dataset/test/Cat /content/
    OIDv4_ToolKit/OID/Dataset/test/valid/
54 !mv /content/OIDv4_ToolKit/OID/Dataset/train/Cat /content/
    OIDv4_ToolKit/OID/Dataset/train/obj
55
56 ##06- Copiar o arquivo yolov4.cfg para o drive , editar o arquivo
    copiado e salvar na pasta darknet/cfg:
57
58 !cp /content/darknet/cfg/yolov4.cfg /content/darknet/cfg/
    yolov4_custom.cfg
59
60 cd /content/converter_annotations_yolo
61
62 !python -c "from utils import configcfg; configcfg('/content/
    darknet/cfg/yolov4_custom.cfg', 64, 1)"
63
64 !cp /content/darknet/cfg/yolov4_custom.cfg /content/gdrive/MyDrive/
    ColabNotebooks/CatDetector/yolov4_custom.cfg
65
66 !cp /content/gdrive/MyDrive/ColabNotebooks/CatDetector/
    yolov4_custom.cfg /content/darknet/cfg
67
68 !cp /content/darknet/cfg/yolov4.cfg /content/darknet/cfg/
    yolov4_custom.cfg
69
70 ##07- Configurar outros arquivos para a darknet, salvar no drive e
    copiar para a darknet/data
71
72 cd /content/gdrive/MyDrive/ColabNotebooks/CatDetector/
73
74 !touch obj.names
75 !touch obj.data
76
77 !echo -e 'Cat' > obj.names
78
79 !cat obj.names
80
81 #Se preferir, pode fazer manualmente essas alteracoes no arquivo
82 !echo -e 'classes = 1\ntrain = /content/darknet/data/train.txt\
    nvalid = /content/darknet/data/test.txt\nnames = /content/
    darknet/data/obj.names\nbackup = /content/gdrive/MyDrive/
    ColabNotebooks/CatDetector/backupWeights/' > obj.data
83
84 !cat obj.data
85
86 !cp /content/darknet/data/obj.names /content/gdrive/MyDrive/
    ColabNotebooks/CatDetector/obj.names
87 !cp /content/darknet/data/obj.data /content/gdrive/MyDrive/
    ColabNotebooks/CatDetector/obj.data
88
89 ##08- Copiar as imagens para a pasta darknet/data e gerar arquivos
    txt com os caminhos de todas as imagens

```

```

90
91 !cp -r /content/OIDv4_ToolKit/OID/Dataset/test/valid /content/
   darknet/data
92 !cp -r /content/OIDv4_ToolKit/OID/Dataset/train/obj /content/
   darknet/data
93
94 !rm /content/darknet/data/obj/Label/*
95 !rm /content/darknet/data/valid/Label/*
96
97 !cp /content/converter_annotations_yolo/gerar_test.py /content/
   darknet/
98 !cp /content/converter_annotations_yolo/gerar_train.py /content/
   darknet/
99
100 cd /content/darknet/
101
102 !python gerar_test.py
103
104 !python gerar_train.py
105
106 ##09- Compilar a darknet
107
108 cd /content/darknet
109
110 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
111 !sed -i 's/GPU=0/GPU=1/' Makefile
112 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
113
114 !make
115
116 ##10- Baixar pesos pre-treinados (transfer learning)
117
118 !wget https://github.com/AlexeyAB/darknet/releases/download/
   darknet_yolo_v3_optimal/yolov4.weights
119
120 !wget https://github.com/AlexeyAB/darknet/releases/download/
   darknet_yolo_v3_optimal/yolov4.conv.137
121
122 ls
123
124 ##11- Testar o funcionamento da darknet
125
126 !./darknet detect cfg/yolov4.cfg yolov4.weights data/person.jpg
127
128 import cv2
129 import matplotlib.pyplot as plt
130 import os
131
132 def mostrar(path):
133     img = cv2.imread(path)
134     fig = plt.gcf()
135     fig.set_size_inches(18, 10)
136     plt.axis("off")
137     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
138     plt.show()
139
140 def detectar(imagem):
141     os.system(f"cd /content/darknet && ./darknet detector test data/obj
   .data cfg/yolov4_custom.cfg yolov4_custom_last.weights {imagem}"
   )

```

```

142 mostrar('predictions.jpg')
143
144 mostrar('/content/darknet/predictions.jpg')
145
146 ##12- Mao na massa: Inicio do treinamento - OBS: Quantidade minima
      de epocas = 2000 * numero de classes
147
148 !cp /content/gdrive/MyDrive/ColabNotebooks/CatDetector/
      yolov4_custom_last.weights ./
149
150 #na primeira vez inicie o treinamento a partir dos pesos pre-
      treinados baixados do github,
151 # os pesos recalculados sao gerados a cada 100 epocas
152 #se necessario interromper o treinamento antes do termino, voce
      deve retoma-lo a partir dos
153 # novos pesos calculados ate a ultima epoca antes da interrupcao
154
155 #iniciano o treinamento
156 !./darknet detector train data/obj.data cfg/yolov4_custom.cfg
      yolov4.weights yolov4.conv.137 -dont_show -map
157
158 #continuando a treinamento apos uma pausa
159 !./darknet detector train data/obj.data cfg/yolov4_custom.cfg
      yolov4_custom_last.weights -dont_show -map
160
161 ##13- Verificando o map (Mean Average Precision)
162
163 # atentar-se para o resultado da variavel ap, quanto mais proximo
      de 100% melhor
164 !./darknet detector map data/obj.data cfg/yolov4_custom.cfg
      yolov4_custom_last.weights
165
166 ##11- Testar o funcionamento da darknet
167
168 !cp /content/gdrive/MyDrive/ColabNotebooks/CatDetector/tito1.jpg ./
169 !cp /content/gdrive/MyDrive/ColabNotebooks/CatDetector/tito2.jpg ./
170 !cp /content/gdrive/MyDrive/ColabNotebooks/CatDetector/tito3.jpg ./
171
172
173 # usa a base de dados coco.names para a identificacao da classe
174 !./darknet detect cfg/yolov4_custom.cfg yolov4_custom_last.weights
      tito3.jpg
175
176 # usa a base de dados que voce criou
177 !./darknet detector test data/obj.data cfg/yolov4_custom.cfg
      yolov4_custom_last.weights tito1.jpg
178
179 #funcao que facilita a deteccao
180 detectar("/content/darknet/tito1.jpg")
181

```

Listing 3: Treinando a rede neural.

Observações: Atenção no momento em que estiver configurando o arquivo obj.data (linha 82), mais precisamente, na última linha, que trata-se do caminho para backup, pois esse caminho deve apontar uma pasta no google drive, para que possamos garantir o salvamento adequado dos arquivos de pesos treinados pela rede.

2.4 Detecção

Depois de um longo tempo treinando a rede neural para identificar com precisão o objeto desejado, já podemos testar o funcionamento na prática, para isso rodamos um código bem simples, no mesmo ambiente de execução do treinamento.

Tendo uma imagem para teste da detecção, basta apenas rodar o código a seguir com a imagem de interesse:

```
1 !./darknet detector test /content/darknet/data/obj.data /content/  
  darknet/cfg/yolov4_custom.cfg /content/gdrive/MyDrive/  
  ColabNotebooks/PigDetector/yolov4_custom_last.weights data/dog.  
  jpg  
2
```

Listing 4: Testando o modelo treinado.

Podemos notar que código para teste é bem similar ao código de treino, precisando apenas substituir o argumento de "train" para "test" e adicionar o caminho da imagem de detecção no final do código.

O **resultado da detecção** é salvo automaticamente em um caminho específico. Para visualizar a predição basta procurar pela imagem **predictions.jpg** que é salva na pasta da darknet.

3 Detecção de Objetos em imagen - YOLO, darknet e OpenCV

Também é possível utilizar o OpenCV para fazermos as detecções, o código é um pouco mais trabalhoso, o resultado é o mesmo quando comparado a detecção. Porém podemos extrair mais informações da detecção.

3.1 Detecção Simples

Link de acesso ao código fonte: [DeteccaoComYoloOpencv](#).

```
1 # Detectando objetos com YOLO v4 - implementacao com OpenCV  
2  
3 ## Etapa 1 - Importando as bibliotecas  
4  
5 import cv2  
6 print(cv2.__version__)  
7  
8 #!pip install opencv-python==4.4.0.40  
9  
10 import cv2  
11 import numpy as np  
12 import time  
13 import os  
14 import matplotlib.pyplot as plt  
15 from google.colab.patches import cv2_imshow  
16 import zipfile  
17 print(cv2.__version__)  
18  
19 ## Etapa 2 - Conectando com o Google Drive  
20  
21 from google.colab import drive  
22 drive.mount('/content/Drive')  
23
```

```

24 ## Etapa 3 - Carregando os arquivos do modelo treinado
25
26 path = '/content/Drive/MyDrive/ColabNotebooks/Curso-
    DeteccaoDeObjetosYolo/DataBase-recursos-curso-Yolo/YOLO/
    modelo_YOLOv4.zip'
27 zip_object = zipfile.ZipFile(file=path, mode='r')
28 zip_object.extractall('./')
29 zip_object.close()
30
31 labels_path = os.path.sep.join(['/content/cfg', 'coco.names'])
32 labels_path
33
34 LABELS = open(labels_path).read().strip().split('\n')
35 print(LABELS)
36
37 len(LABELS)
38
39 weights_path = os.path.sep.join(['/content', 'yolov4.weights'])
40 config_path = os.path.sep.join(['/content/cfg', 'yolov4.cfg'])
41 weights_path, config_path
42
43 net = cv2.dnn.readNet(config_path, weights_path)
44
45 net
46
47 ## Etapa 4 - Definindo mais configuracoes para a deteccao
48
49 COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype='
    uint8')
50 print(COLORS)
51
52 layerNames = net.getLayerNames()
53 print('Todas as camadas')
54 print(layerNames)
55 print('Total de camadas: ' + str(len(layerNames)))
56
57 print('Indices das camadas de saida')
58 print(net.getUnconnectedOutLayers())
59 print('Camadas de saida')
60 ln = net.getUnconnectedOutLayersNames()
61 print(ln)
62
63 ## Etapa 5 - Carregando a imagem onde sera feita a deteccao
64
65 def mostrar(img):
66 fig = plt.gcf()
67 fig.set_size_inches(16,10)
68 plt.axis('off')
69 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
70 plt.show()
71
72 parent_dir = os.path.dirname(path)
73 path, parent_dir
74
75 import subprocess
76
77 images_path = os.path.join(parent_dir, 'imagens')
78 result = subprocess.run(['ls', images_path], capture_output=True,
    text=True)
79 print(result.stdout)

```

```

80 images_path = [os.path.join(images_path, f) for f in os.listdir(
    images_path) if f.lower().endswith('.jpg')]
81 images_path
82
83 for img in images_path:
84 mostrar(cv2.imread(img))
85
86 imagem = cv2.imread(images_path[0])
87
88 type(imagem)
89
90 print(imagem)
91
92 imagem_cp = imagem.copy()
93
94 imagem.shape
95
96 (H, W) = imagem.shape[:2]
97 print('Altura: ' + str(H) + '\nLargura: ' + str(W))
98
99 ## Etapa 6 - Processando a imagem de entrada
100
101 inicio = time.time()
102
103 blob = cv2.dnn.blobFromImage(imagem, 1 / 255.0, (416, 416), swapRB
    = True, crop = False)
104 net.setInput(blob)
105 layer_outputs = net.forward(ln)
106
107 termino = time.time()
108 print('YOLO levou {:.2f} segundos'.format(termino - inicio))
109
110 ## Etapa 7 - Definindo as variaveis
111
112 threshold = 0.5
113 threshold_NMS = 0.3
114 caixas = []
115 confiancas = []
116 IDclasses = []
117
118 ## Etapa 8 - Realizando a predicao
119
120 len(layer_outputs)
121
122 layer_outputs[0], len(layer_outputs[0])
123
124 layer_outputs[1]
125
126 layer_outputs[2], layer_outputs[2][0], len(layer_outputs[2][0])
127
128 print(LABELS)
129
130 LABELS[16]
131
132 np.argmax(np.array([0.10, 0.80, 0.10]))
133
134 teste = np.array([0.10, 0.80, 0.10])
135
136 teste[1]
137

```

```

138 for output in layer_outputs:
139 for detection in output:
140 scores = detection[5:]
141 classeID = np.argmax(scores)
142 confianca = scores[classeID]
143 if confianca > threshold:
144 print('scores: ' + str(scores))
145 print('classe mais provavel: ' + str(classeID))
146 print('confianca: ' + str(confianca))
147
148 caixa = detection[0:4] * np.array([W, H, W, H])
149 (centerX, centerY, width, height) = caixa.astype('int')
150
151 x = int(centerX - (width / 2))
152 y = int(centerY - (height / 2))
153
154 caixas.append([x, y, int(width), int(height)])
155 confiancas.append(float(confianca))
156 IDclasses.append(classeID)
157
158 print(caixas), len(caixas)
159
160 print(confiancas), len(confiancas)
161
162 print(IDclasses), len(IDclasses)
163
164 ## Etapa 9 - Aplicando a Non-Maxima Suppression
165
166 objs = cv2.dnn.NMSBoxes(caixas, confiancas, threshold,
167                          threshold_NMS)
168
169 objs
170
171 print(objs.flatten())
172
173 confiancas[7], confiancas[15], confiancas[11]
174
175 confiancas[0]
176
177 ## Etapa 10 - Mostrando o resultado da deteccao na imagem
178
179 if len(objs) > 0:
180 for i in objs.flatten():
181 (x, y) = (caixas[i][0], caixas[i][1])
182 (w, h) = (caixas[i][2], caixas[i][3])
183
184 objeto = imagem_cp[y:y + h, x:x + w]
185 cv2_imshow(objeto)
186
187 cor = [int(c) for c in COLORS[IDclasses[i]]]
188
189 cv2.rectangle(imagem, (x, y), (x + w, y + h), cor, 2)
190 texto = "{}: {:.4f}".format(LABELS[IDclasses[i]], confiancas[i])
191 cv2.putText(imagem, texto, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
192            0.5, cor, 2)
193
194 mostrar(imagem)
195
196 cv2.imwrite('resultado.jpg', imagem)

```

Listing 5: Detecção de imagem com OpenCV e YOLO.

3.2 Contagem de Objetos

Através da detecção de objetos com o uso do OpenCV, abrimos algumas possibilidades além da detecção simples dos objetos na imagem, também conseguimos por exemplo fazer a contagem de cada objeto encontrado. Podemos passar uma lista de imagens e fazermos a detecção de cada uma delas enquanto atualizamos a quantidade de objetos encontrados durante as detecções. Link de acesso ao código fonte: **DeteccaoComYoloOpencv**.

```

1 # Detectando objetos com YOLO e OpenCV - Explorando mais opcoes
2
3 ## Etapa 1 - Importando as bibliotecas
4
5 import cv2
6 print(cv2.__version__)
7
8 !pip install opencv-python==4.4.0.40
9
10 import cv2
11 import numpy as np
12 import time
13 import os
14 import matplotlib.pyplot as plt
15 from google.colab.patches import cv2_imshow
16 import zipfile
17 print(cv2.__version__)
18
19 ## Etapa 2 - Conectando com o Google Drive
20
21 from google.colab import drive
22 drive.mount('/content/gdrive')
23
24 ## Etapa 3 - Carregando os arquivos do modelo treinado
25
26 path = '/content/gdrive/MyDrive/ColabNotebooks/Curso-
      DeteccaoDeObjetosYolo/DataBase-recursos-curso-Yolo/YOLO/
      modelo_YOLOv4.zip'
27 zip_object = zipfile.ZipFile(file=path, mode='r')
28 zip_object.extractall('./')
29 zip_object.close()
30
31 labelsPath = os.path.sep.join(['/content/cfg', "coco.names"])
32 LABELS = open(labelsPath).read().strip().split("\n")
33 LABELS
34
35 weightsPath = os.path.sep.join(['/content/', "yolov4.weights"])
36 configPath = os.path.sep.join(['/content/cfg', "yolov4.cfg"])
37
38 net = cv2.dnn.readNet(configPath, weightsPath)
39
40 ## Etapa 4 - Definindo mais configuracoes para a deteccao
41
42 np.random.seed(42)

```



```

43 COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype="
    uint8")
44
45 all_layers = net.getLayerNames()
46 print("Todas as camadas (layers):")
47 print(all_layers)
48 print("Total: " + str(len(all_layers)))
49
50 ln = net.getUnconnectedOutLayersNames()
51 print(f"Camadas de saida: {ln}")
52 print(f"Indices das camadas de saida: {net.getUnconnectedOutLayers
    ()}")
53
54
55 ## Etapa 5 - Carregando a imagem onde sera feita a deteccao
56
57 def mostrar(img):
58     fig = plt.gcf()
59     fig.set_size_inches(16, 10)
60     plt.axis("off")
61     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
62     plt.show()
63
64 images_path = os.listdir(os.path.join(os.path.dirname(path), '
    imagens'))
65 images = [os.path.join(os.path.join(os.path.dirname(path), 'imagens
    '), i) for i in images_path if i.lower().endswith('.jpg')]
66 images
67
68 imagem = cv2.imread(images[5])
69 mostrar(imagem)
70 imagem_cp = imagem.copy()
71 (H, W) = imagem.shape[:2]
72 print("Altura: " + str(H) + "\nLargura: " + str(W))
73
74 ## Etapa 6 - Processando a imagem de entrada
75
76 ### Redimensionamento da imagem (opcional)
77
78 imagem.shape
79
80 proporcao = imagem.shape[1] / imagem.shape[0]
81 proporcao
82
83 def redimensionar(imagem, largura_maxima = 600):
84     if imagem.shape[1] > largura_maxima:
85         proporcao = imagem.shape[1] / imagem.shape[0]
86         imagem_largura = largura_maxima
87         imagem_altura = int(imagem_largura / proporcao)
88     else:
89         imagem_largura = imagem.shape[1]
90         imagem_altura = imagem.shape[0]
91
92     imagem = cv2.resize(imagem, (imagem_largura, imagem_altura))
93     return imagem
94
95 imagem = redimensionar(imagem)
96
97 mostrar(imagem)
98 (H, W) = imagem.shape[:2]

```

```

99 imagem_cp = imagem.copy()
100 print('Altura: ' + str(H) + '\nLargura: ' + str(W))
101
102 ### Construindo o blob da imagem
103
104 def blob_imagem(net, imagem, mostrar_texto=True):
105     inicio = time.time()
106
107     blob = cv2.dnn.blobFromImage(imagem, 1 / 255.0, (416, 416), swapRB=
        True, crop=False)
108     net.setInput(blob)
109     layerOutputs = net.forward(ln)
110
111     termino = time.time()
112
113     if mostrar_texto:
114         print("YOLO levou {:.2f} segundos".format(termino - inicio))
115
116     return net, imagem, layerOutputs
117
118 net, imagem, layerOutputs = blob_imagem(net, imagem)
119
120 ## Etapa 7 - Definindo as variaveis
121
122 _threshold = 0.5
123 _threshold_NMS = 0.3
124 caixas = []
125 confiancas = []
126 IDclasses = []
127
128 ## Etapa 8 - Realizando a predicao
129
130 def deteccoes(detection, _threshold, caixas, confiancas, IDclasses)
    :
131     scores = detection[5:]
132     classeID = np.argmax(scores)
133     confianca = scores[classeID]
134
135     if confianca > _threshold:
136         caixa = detection[0:4] * np.array([W, H, W, H])
137         (centerX, centerY, width, height) = caixa.astype("int")
138         x = int(centerX - (width / 2))
139         y = int(centerY - (height / 2))
140
141         caixas.append([x, y, int(width), int(height)])
142         confiancas.append(float(confianca))
143         IDclasses.append(classeID)
144
145     return caixas, confiancas, IDclasses
146
147 for output in layerOutputs:
148     for detection in output:
149         caixas, confiancas, IDclasses = deteccoes(detection, _threshold,
            caixas, confiancas, IDclasses)
150
151 print(caixas)
152 print(confiancas)
153 print(IDclasses)
154
155 len(caixas)

```

```

156
157 ## Etapa 9 - Aplicando a Non-Maxima Suppression
158
159 objs = cv2.dnn.NMSBoxes(caixas, confiancas, _threshold,
    _threshold_NMS)
160
161 print("Objetos detectados: " + str(len(objs)))
162
163 ## Etapa 10 - Mostrando o resultado da deteccao na imagem
164
165 def check_negativo(n):
166     if (n < 0):
167         return 0
168     else:
169         return n
170
171 def funcoes_imagem(imagem, i, confiancas, caixas, COLORS, LABELS,
    mostrar_texto=True):
172     (x, y) = (caixas[i][0], caixas[i][1])
173     (w, h) = (caixas[i][2], caixas[i][3])
174     cor = [int(c) for c in COLORS[IDclasses[i]]]
175     cv2.rectangle(imagem, (x, y), (x + w, y + h), cor, 2)
176     texto = "{}: {:.4f}".format(LABELS[IDclasses[i]], confiancas[i])
177     if mostrar_texto:
178         print("> " + texto)
179         print(x,y,w,h)
180     cv2.putText(imagem, texto, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
        0.5, cor, 2)
181
182     return imagem,x,y,w,h
183
184 if len(objs) > 0:
185     for i in objs.flatten():
186         imagem, x, y, w, h = funcoes_imagem(imagem, i, confiancas, caixas,
            COLORS, LABELS)
187         objeto = imagem_cp[y:y + h, x:x + w]
188         cv2.imshow(objeto)
189
190     mostrar(imagem)
191
192 ## Fazendo a deteccao em multiplas imagens de uma vez
193
194 diretorio_fotos = '/content/gdrive/MyDrive/ColabNotebooks/Curso-
    DeteccaoDeObjetosYolo/DataBase-recursos-curso-Yolo/YOLO/
    Atualizacao YOLOv8/fotos_teste'
195 caminhos = [os.path.join(diretorio_fotos, i) for i in os.listdir(
    diretorio_fotos) if i.lower().endswith('.jpg')]
196 caminhos
197
198 for caminho_imagem in caminhos:
199     try:
200         imagem = cv2.imread(caminho_imagem)
201         (H, W) = imagem.shape[:2]
202     except:
203         print('Erro ao carregar a imagem -> ' + caminho_imagem)
204         continue
205
206     imagem_cp = imagem.copy()
207     net, imagem, layer_outputs = blob_imagem(net, imagem)
208

```

```

209 caixas = []
210 confiancas = []
211 IDclasses = []
212
213 for output in layer_outputs:
214     for detection in output:
215         caixas, confiancas, IDclasses = deteccoes(detection, _threshold,
216             caixas, confiancas, IDclasses)
217
218     objs = cv2.dnn.NMSBoxes(caixas, confiancas, _threshold,
219         _threshold_NMS)
220
221     if len(objs) > 0:
222         for i in objs.flatten():
223             imagem, x, y, w, h = funcoes_imagem(imagem, i, confiancas, caixas,
224                 COLORS, LABELS, mostrar_texto=False)
225             objeto = imagem_cp[y:y + h, x:x + w]
226             mostrar(imagem)
227
228             ## Contando quantas vezes algum objeto especifico apareceu em
229             ## multiplas imagens
230
231             diretorio_fotos = "fotos_teste/"
232             caminhos = [os.path.join(diretorio_fotos, f) for f in os.listdir(
233                 diretorio_fotos)]
234             print(caminhos)
235             threshold = 0.5
236             threshold_NMS = 0.3
237
238             contagem = 0
239             contagem_total = 0
240
241             for caminho_imagem in caminhos:
242                 try:
243                     imagem = cv2.imread(caminho_imagem)
244                     (H, W) = imagem.shape[:2]
245                 except:
246                     print('Erro ao carregar a imagem -> ' + caminho_imagem)
247                     continue
248
249                 contagem = 0
250
251                 imagem_cp = imagem.copy()
252                 net, imagem, layer_outputs = blob_imagem(net, imagem)
253
254                 caixas = []
255                 confiancas = []
256                 IDclasses = []
257
258                 for output in layer_outputs:
259                     for detection in output:
260                         caixas, confiancas, IDclasses = deteccoes(detection, threshold,
261                             caixas, confiancas, IDclasses)
262
263                     objs = cv2.dnn.NMSBoxes(caixas, confiancas, threshold,
264                         threshold_NMS)
265
266                     if len(objs) > 0:
267                         for i in objs.flatten():

```

```

262
263 if LABELS[IDclasses[i]] == 'person':
264     contagem += 1
265     contagem_total += 1
266
267 imagem, x, y, w, h = funcoes_imagem(imagem, i, confiancas, caixas,
    COLORS, LABELS, mostrar_texto=False)
268 objeto = imagem_cp[y:y + h, x:x + w]
269
270 print('Pessoas detectadas na imagem ' + str(caminho_imagem) + ': '
    + str(contagem))
271 mostrar(imagem)
272
273 print('Total de pessoas detectadas: ' + str(contagem_total))
274
275 ## Fazendo a deteccao apenas de objetos especificos do modelo
276
277 threshold = 0.5
278 threshold_NMS = 0.3
279
280 classes = ['dog', 'cat']
281 for caminho_imagem in caminhos:
282     try:
283         imagem = cv2.imread(caminho_imagem)
284         (H, W) = imagem.shape[:2]
285     except:
286         print('Erro ao carregar a imagem -> ' + caminho_imagem)
287         continue
288
289     imagem_cp = imagem.copy()
290     net, imagem, layer_outputs = blob_imagem(net, imagem)
291
292     caixas = []
293     confiancas = []
294     IDclasses = []
295
296     for output in layer_outputs:
297         for detection in output:
298             caixas, confiancas, IDclasses = deteccoes(detection, _threshold,
                caixas, confiancas, IDclasses)
299
300     objs = cv2.dnn.NMSBoxes(caixas, confiancas, _threshold,
        _threshold_NMS)
301
302     if len(objs) > 0:
303         for i in objs.flatten():
304             if LABELS[IDclasses[i]] in classes:
305                 imagem, x, y, w, h = funcoes_imagem(imagem, i, confiancas, caixas,
                    COLORS, LABELS, mostrar_texto=False)
306                 objeto = imagem_cp[y:y + h, x:x + w]
307
308                 mostrar(imagem)
309
310     ## Modo mais legivel
311
312     def funcoes_imagem_v2(imagem, i, confiancas, caixas, COLORS, LABELS
        , mostrar_texto=True):
313         (x, y) = (caixas[i][0], caixas[i][1]) # coordenada (x,y) onde
            inicia a caixa da deteccao

```

```

314 (w, h) = (caixas[i][2], caixas[i][3]) # largura e altura em pixels
    da caixa de deteccao
315
316 cor = [int(c) for c in COLORS[IDclasses[i]]]
317
318 fundo = np.full((imagem.shape), (0,0,0), dtype=np.uint8)
319
320 texto = "{}: {:.4f}".format(LABELS[IDclasses[i]], confiancas[i])
321
322 cv2.putText(fundo, texto, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
    0.5, (255,255,255), 2)
323
324 fx,fy,fw,fh = cv2.boundingRect(fundo[:, :, 2])
325
326 cv2.rectangle(imagem, (x, y), (x + w, y + h), cor, 2)
327
328 cv2.rectangle(imagem, (fx, fy), (fx + fw, fy + fh), cor, -1)
329 cv2.rectangle(imagem, (fx, fy), (fx + fw, fy + fh), cor, 3)
330 cv2.putText(imagem, texto, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
    0.5, (0,0,0), 1)
331
332 if mostrar_texto:
333     print("> " + texto)
334     print(x,y,w,h)
335
336 return imagem,x,y,w,h
337
338 for caminho_imagem in caminhos:
339     try:
340         imagem = cv2.imread(caminho_imagem)
341         (H, W) = imagem.shape[:2]
342     except:
343         print('Erro ao carregar a imagem -> ' + caminho_imagem)
344         continue
345
346 imagem_cp = imagem.copy()
347 net, imagem, layer_outputs = blob_imagem(net, imagem)
348
349 caixas = []
350 confiancas = []
351 IDclasses = []
352
353 for output in layer_outputs:
354     for detection in output:
355         caixas, confiancas, IDclasses = deteccoes(detection, _threshold,
            caixas, confiancas, IDclasses)
356
357 objs = cv2.dnn.NMSBoxes(caixas, confiancas, _threshold,
    _threshold_NMS)
358
359 if len(objs) > 0:
360     for i in objs.flatten():
361         imagem, x, y, w, h = funcoes_imagem_v2(imagem, i, confiancas,
            caixas, COLORS, LABELS, mostrar_texto=False)
362         objeto = imagem_cp[y:y + h, x:x + w]
363
364 mostrar(imagem)
365

```

Listing 6: Contagem de objetos específicos.

4 Detecção de Objetos em vídeo - YOLO e darknet

O resultado da detecção em vídeo é também um vídeo, porém com as caixas delimitadoras para cada objeto detectado. Esse resultado é salvo com a extensão .avi. Assim como para imagens, no caso de vídeos, o resultado é salvo em um caminho pré-definido e a cada nova detecção uma pasta contendo o vídeo resuالتado é criada nesse caminho. Link e acesso ao código fonte: **DeteccaoEmVideoComYoloDarknet**.

```
1 #Deteccao em videos - darknet
2
3 ##1- Download da darknet
4
5 !git clone https://github.com/AlexeyAB/darknet.git
6
7 cd darknet
8
9 ##2- Compilar a darknet para uso da GPU
10
11 ###Verificando o uso da GPU:
12
13
14 import tensorflow as tf
15 device = tf.test.gpu_device_name()
16
17 if device:
18     print(f"GPU habilitada: {device}")
19 else:
20     print(f"Utilizando apenas CPU")
21
22 ###Compilando a darknet
23
24 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
25 !sed -i 's/GPU=0/GPU=1/' Makefile
26 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
27
28 !make
29
30 ##3- Carregar pesos preh-treinados
31
32 from google.colab import drive
33 import zipfile
34 drive.mount('/content/gdrive')
35
36 cd /content
37
38 path = "/content/gdrive/MyDrive/yolov4/modelo_YOLOv4.zip"
39 zip_obj = zipfile.ZipFile(file=path, mode='r')
40 zip_obj.extractall('./')
41 zip_obj.close()
42
43 ##Carregando o video
44
45 ###De uma URL
46
47 !wget https://github.com/gabevr/yolo/raw/master/videos/
    video_teste02.mp4
```

```

48
49 ###Do google drive
50
51 !cp /content/gdrive/MyDrive/yolov4/video01.mp4 ./
52
53 ##4- Realizando a deteccao em video
54
55 cd /content/darknet/
56
57 !./darknet detector demo cfg/coco.data cfg/yolov4.cfg /content/
    yolov4.weights -dont_show /content/video01.mp4 -i 0 -
    out_filename resultado.avi
58
59 ###Verificando o tamanho do arquivo resultado
60
61 !du -h resultado.avi
62
63 ###Copiando o resultado para o drive
64
65 !cp ./resultado.avi /content/gdrive/MyDrive/yolov4/resultados/
    resultado1.avi
66
67 ##5- Especificando um threshold
68
69 !cp /content/gdrive/MyDrive/yolov4/video_rua01.mp4 /content/
70
71 !./darknet detector demo cfg/coco.data cfg/yolov4.cfg /content/
    yolov4.weights -dont_show /content/video_rua01.mp4 -i 0 -
    out_filename resultado2.mp4 -thresh 0.3
72
73 !cp ./resultado2.mp4 /content/gdrive/MyDrive/yolov4/resultados/
    resultado2.mp4
74

```

Listing 7: Detecção em vídeo com Yolo e darknet.

5 Detecção de Objetos em vídeo - YOLO, darknet e OpenCV

6 Guia - Fundamentos OpenCV

6.1 Introdução

Link de acesso ao conteúdo completo: **Learn-opencv-in-3-hours**.

Resoluções comuns de imagens e vídeos (em pixels):





| | |
|--|--|
|  VGA: 640 x 480 |  FHD: 1920 x 1080 |
|  HD: 1280 x 720 |  4K: 3840 x 2160 |

Imagem Binária: aquela em que cada pixel pode ter apenas um de dois valores possíveis: preto (geralmente representado pelo valor 0) ou branco (geralmente representado pelo valor 1).

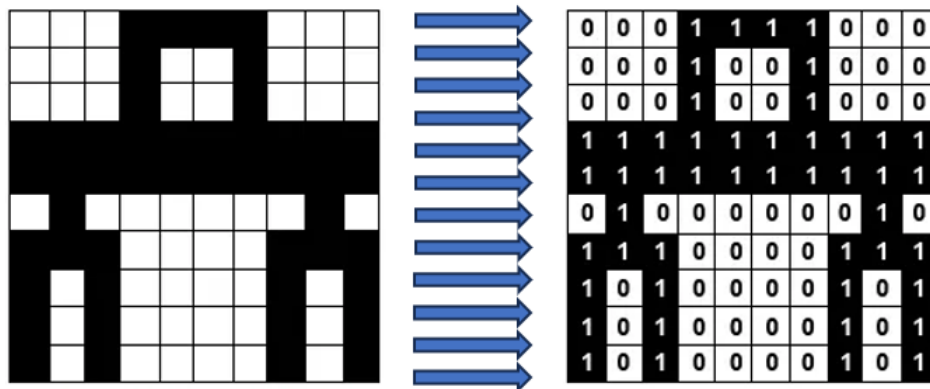


Figura 4: Imagem Binária.

Download Python: para instalar o python no pc é simples basta pesquisar no seu navegador por python download ou então acessar o link **download-python** e escolher a versão estável mais recente, conforme o sistema operacional da sua máquina.

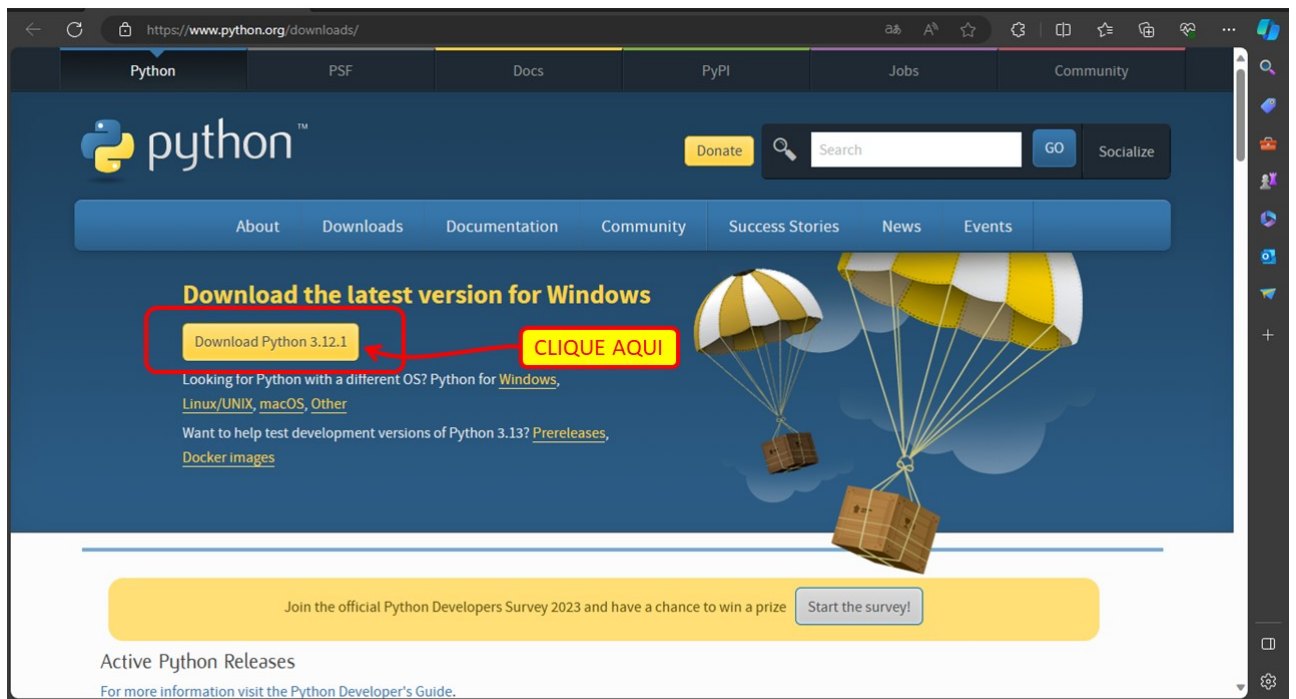


Figura 5: Download python.

6.1.1 *imread()*, *imshow()*, *waitKey()*

Para ler uma imagem com opencv usamos a função *imread()*. Essa função precisa apenas do caminho para a imagem, assim a leitura ocorrerá adequadamente. O caminho pode ser tanto relativo quanto absoluto.

Após a leitura da imagem, já é possível mostrarmos ela na tela, para isso usamos a função *imshow()*. Porém apenas com essas duas funções ainda não teremos o resultado esperado, para que a imagem seja mostrada na tela por tempo indeterminado, também precisamos usar a função *waitKey()*, assim a imagem não irá aparecer e sumir repentinamente.

Como padrão, utilize o número 0 para o delay de milissegundos na função *waitKey()*.

```
1 import cv2 # importacao da biblioteca opencv
2
3 # lendo as imagens
4 imagem_caminho_relativo = cv2.imread("imagens/saasweeklogo.png")
5 imagem_caminho_absoluto = cv2.imread("C:/Users/family/Documents/
    CodigosPython/opencv/imagens/user_1.jpg")
6
7 # mostrando as imagens lidas
8 cv2.imshow("imagem caminho relativo", imagem_caminho_relativo)
9 cv2.imshow("imagem caminho absoluto", imagem_caminho_absoluto)
10 cv2.waitKey(0)
11
12
```

Listing 8: Lendo e visualizando imagens com OpenCV.

6.1.2 VideoCapture()

Para a leitura de vídeos também uma função específica para isso, nesse caso usamos a função VideoCapture() que pode receber como argumento, um número inteiro quando queremos referenciar uma câmera e fazemos uma apresentação em tempo real, ou então podemos passar o caminho de um vídeo como parâmetro. No caso de usarmos um número inteiro, geralmente o número 0 referencia a primeira webcam atrelada a sua máquina.

O delay de 1 milissegundo(ms) fará com que cada frame seja apresentando no intervalo de 1 ms.

Uma boa prática na visualização de vídeos com OpenCV é termos no final do código duas linhas bem importantes, a primeira que vai liberar a câmera que estava sendo utilizada anteriormente no código e a segunda vai assegurar de destruir qualquer janela que eventualmente foi criada pelo OpenCV.

```
1 import cv2 # importacao da biblioteca opencv
2
3 # lendo videos
4 captura = cv2.VideoCapture("videos/joao_balaio.mp4")
5
6 # mostrando cada frame do video
7 while True:
8     sucesso, frame = captura.read()
9     cv2.imshow("Video", frame)
10
11     # intervalo de 1 ms entre frame, tecla 'q'
12     if cv2.waitKey(1) == ord('q'):
13         break
14 captura.release()
15 cv2.destroyAllWindows()
16
```

Listing 9: Lendo e visualizando videos com OpenCV.

6.1.3 set()

Para a leitura de webcams o código é quase idêntico, se você estiver utilizando um notebook com webcam ou então um computador que possua apenas uma webcam, para acessar essa webcaam usamos o numeral 0 para a captura do vídeo.

Também podemos alterar as dimensões da imagem/vídeo a partir da função *set()* na qual inserimos dois parâmetros, o primeiro indica o que queremos alterar, como altura ou largura e o segundo é o novo valor.

```
1 import cv2 # importacao da biblioteca opencv
2
3 # lendo webcam
4 captura = cv2.VideoCapture(0)
5
6 # Alterando largura da imagem
7 captura.set(3, 640)
8
9 # Alterando altura da imagem
10 captura.set(4, 480)
11
12 # mostrando cada frame do video
13 while True:
14     sucesso, frame = captura.read()
15     cv2.imshow("Video", frame)
16
17     # intervalo de 1 ms entre frame, tecla 'q'
18     if cv2.waitKey(1) == ord('q'):
19         break
20 captura.release()
21 cv2.destroyAllWindows()
22
```

Listing 10: Lendo webcams com OpenCV.

6.1.4 *flip()*

Caso o vídeo capturado pela webcam esteja espelhado, para corrigir esse problema é bem simples, usamos a função *flip()*. Essa função recebe um único parâmetro que podem ser os valores -1, 0 ou 1. Que atuam da seguinte forma:

| Valor | Tipo de Espelhamento | Sintaxe |
|-------|-----------------------|-----------|
| 1 | Horizontal | .flip(1) |
| 0 | Vertical | .flip(0) |
| -1 | Horizontal e Vertical | .flip(-1) |

Tabela 1: Descrição dos Espelhamentos no OpenCV.

6.2 Funções Básicas

6.2.1 Convertendo uma imagem para a escala de cinza

A conversão de uma imagem colorida para outra em uma escala de cinza é muito comum, para isso utilizamos a função *cvtColor()* e o método *COLOR_BGR2GRAY*. A aplicação é bem simples:

```
1 import cv2 # importacao da biblioteca opencv
2
3 # lendo imagem
4 imagem = cv2.imread("imagens/user_1.jpg")
5
6 # convertendo a imagem para a escala de cinza
7 imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
```

```

8
9 # mostrando a imagem lida
10 cv2.imshow("Imagem Cinza", imagem_cinza)
11 cv2.waitKey(0)
12

```

Listing 11: Imagem convertida para escala de cinza.

6.2.2 Filtro Gaussiano, eliminando ruídos de imagens

Esse recurso é aplicado para a suavização de ruídos presentes em imagens, eliminando detalhes que as vezes não são desejáveis. Também é muito utilizado no opencv. Para criarmos uma imagem com esse filtro usamos a função *GaussianBlur()*. Essa função requer alguns parâmetros:

| Parâmetro | Descrição |
|------------|---|
| src | A imagem de entrada. |
| kernelSize | Tamanho do kernel (ou matriz) do filtro gaussiano. Deve ser um número ímpar positivo. |
| sigmaX | Desvio padrão ao longo do eixo x (horizontal). |
| dst | (Opcional) Imagem de saída. |
| sigmaY | (Opcional) Desvio padrão ao longo do eixo y (vertical). |
| borderType | (Opcional) Método de preenchimento de borda. |

Tabela 2: Parâmetros da função GaussianBlur em OpenCV.

```

1 import cv2 # importacao da biblioteca opencv
2
3 # lendo imagem
4 imagem = cv2.imread("imagens/user_1.jpg")
5
6 # convertendo a imagem para a escala de cinza
7 imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
8
9 # aplicando o filtro gaussiano
10 imagem_blur = cv2.GaussianBlur(imagem_cinza, (7, 7), 0)
11
12 # mostrando a imagem lida
13 cv2.imshow("Imagem Cinza", imagem_cinza)
14 cv2.imshow("Imagem Original", imagem)
15 cv2.imshow("Imagem Blur", imagem_blur)
16 cv2.waitKey(0)
17

```

Listing 12: Filtro Gaussiano (blur).

6.2.3 Filtro Canny, detecção de bordas

A ideia por trás do algoritmo Canny é encontrar áreas na imagem onde ocorre uma mudança abrupta de intensidade, o que geralmente indica a presença de uma

borda. No OpenCV usamos a função *Canny()* para obtermos uma imagem com esse tipo de filtro. Alguns parâmetros também são necessários:

| Parâmetro | Descrição |
|--------------|--|
| image | A imagem de entrada (pode ser em tons de cinza). |
| threshold1 | Primeiro limiar para o detector de bordas. |
| threshold2 | Segundo limiar para o detector de bordas. |
| edges | A imagem de saída que contém bordas após a detecção de Canny. |
| apertureSize | Tamanho do kernel do operador Sobel (opcional). |
| L2gradient | Um flag booleano indicando se usar a norma L2 para o gradiente (opcional). |

Tabela 3: Parâmetros da função **Canny** em OpenCV.

Limiar Inferior (Threshold1): Se você aumentar o limiar inferior, será mais difícil para os pixels serem classificados como bordas, resultando em menos detalhes detectados. Se você diminuir o limiar inferior, mais pixels serão considerados como bordas, aumentando a sensibilidade, mas também aumentando a probabilidade de falsos positivos.

Limiar Superior (Threshold2): Aumentar o limiar superior tornará a detecção de bordas mais restritiva, mantendo apenas as bordas mais fortes. Reduzir o limiar superior permitirá que bordas mais fracas sejam consideradas como parte da borda. No exemplo a seguir implementamos o filtro de Canny:

```
1 import cv2 # importacao da biblioteca opencv
2 import numpy as np
3
4 # lendo imagem
5 imagem = cv2.imread("imagens/user_1.jpg")
6
7 # convertendo a imagem para a escala de cinza
8 imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
9
10 # aplicando o filtro gaussiano
11 imagem_blur = cv2.GaussianBlur(imagem_cinza, (7, 7), 0)
12
13 # aplicando o filtro de canny, gera uma imagem preta com linhas
    brancas
14 imagem_canny = cv2.Canny(imagem, 100, 100)
15
16 # mostrando a imagem lida
17 cv2.imshow("Imagem Cinza", imagem_cinza)
18 cv2.imshow("Imagem Original", imagem)
19 cv2.imshow("Imagem Blur", imagem_blur)
20 cv2.imshow("Imagem Canny", imagem_canny)
21
22 cv2.waitKey(0)
23
```

Listing 13: Filtro de Canny no OpenCV.

6.2.4 Filtro para aumentar o realce de bordas - dilate

Uma técnica para evidenciar linhas com falhas ou fracas é aplicar o filtro de dilatação, assim as linhas ficam mais largas fazendo com que essas falhas desapareçam, dando um aspecto de linhas mais contínuas e mais largas.

Para isso usamos a função *dilate()*, nesse caso um dos parâmetros da função é o kernel, ou seja, a mascara que será aplicada na imagem, esse kernel deve ser uma matriz de números 1, que pode ser gerada pelo pacote **numpy** . Veja a seguir:

```
1 import cv2 # importacao da biblioteca opencv
2 import numpy as np
3
4 # lendo imagem
5 imagem = cv2.imread("imagens/user_1.jpg")
6
7 # convertendo a imagem para a escala de cinza
8 imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
9
10 # aplicando o filtro gaussiano
11 imagem_blur = cv2.GaussianBlur(imagem_cinza, (7, 7), 0)
12
13 # aplicando o filtro de canny, gera uma imagem preta com linhas
    brancas
14 imagem_canny = cv2.Canny(imagem, 100, 100)
15
16 # aplicando o filtro de dilatacao
17 mascara = np.ones((5,5), np.uint8)
18 imagem_dilatada = cv2.dilate(imagem_canny, mascara, iterations=1)
19
20
21 # mostrando a imagem lida
22 cv2.imshow("Imagem Cinza", imagem_cinza)
23 cv2.imshow("Imagem Original", imagem)
24 cv2.imshow("Imagem Blur", imagem_blur)
25 cv2.imshow("Imagem Canny", imagem_canny)
26 cv2.imshow("Imagem Dilatada", imagem_dilatada)
27 cv2.waitKey(0)
28
```

Listing 14: Filtro de Dilatação no OpenCV.

Para verificarmos o resultado da operação podemos comparar uma imagem com o filtro de Canny com uma imagem com o filtro de Dilatação:



Figura 6: Diferença entre o Filtro de Canny e o Filtro de Dilatação.

Os parâmetros dessa função são os seguintes: Quanto mais iterações houver,

| Parâmetro | Descrição |
|-------------------------------------|---|
| <code>src</code> | A imagem de entrada. |
| <code>kernel</code> | O elemento estruturante (kernel) usado para a operação de dilatação. |
| <code>dst</code> (opcional) | A imagem de saída (pode ser a mesma que a imagem de entrada). |
| <code>anchor</code> (opcional) | Posição do ponto central do kernel. O valor padrão é <code>(-1, -1)</code> , que significa o centro do kernel. |
| <code>iterations</code> (opcional) | Número de vezes que a dilatação é aplicada. O valor padrão é 1. |
| <code>borderType</code> (opcional) | Método de preenchimento de borda. O valor padrão é <code>cv2.BORDER_CONSTANT</code> . |
| <code>borderValue</code> (opcional) | Valor a ser usado para preenchimento de borda quando <code>borderType</code> é <code>cv2.BORDER_CONSTANT</code> . O valor padrão é 0. |

Tabela 4: Parâmetros da função `dilate` em OpenCV.

mais largas as linhas ficarão.

6.2.5 Filtro para diminuir o realce de bordas - `erode`

Esse filtro é exatamente o oposto do filtro de dilatação, ou seja, as linhas ficam mais finas através desse recurso, isso serve para eliminarmos linhas indesejadas na imagem, também é comumente usada em conjunto com outros filtros para que depois de dilatarmos uma imagem e reduzirmos suas linhas com falhas então possamos voltar a afiná-las para voltarmos a imagem anterior agora com mais detalhes.

No OpenCV para aplicar tal filtro usamos a função `erode()` que também possuem alguns parâmetros importantes.

```

1 import cv2 # importacao da biblioteca opencv
2 import numpy as np
3
4 # lendo imagem
5 imagem = cv2.imread("imagens/user_1.jpg")
6
7 # convertendo a imagem para a escala de cinza
8 imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
9
10 # aplicando o filtro gaussiano
11 imagem_blur = cv2.GaussianBlur(imagem_cinza, (7, 7), 0)
12
13 # aplicando o filtro de canny, gera uma imagem preta com linhas
    brancas
14 imagem_canny = cv2.Canny(imagem, 100, 100)
15
16 # aplicando o filtro de dilatacao
17 mascara = np.ones((5,5), np.uint8)
18 imagem_dilatada = cv2.dilate(imagem_canny, mascara, iterations=1)
19
20 # aplicando o filtro de erosao
21 imagem_erosao = cv2.erode(imagem_dilatada, mascara, iterations=1)
22
23 # mostrando a imagem lida
24 cv2.imshow("Imagem Cinza", imagem_cinza)
25 cv2.imshow("Imagem Original", imagem)
26 cv2.imshow("Imagem Blur", imagem_blur)
27 cv2.imshow("Imagem Canny", imagem_canny)
28 cv2.imshow("Imagem Dilatada", imagem_dilatada)
29 cv2.imshow("Imagem Erosao", imagem_erosao)
30 cv2.waitKey(0)
31

```

Listing 15: Filtro de Erosão no OpenCV.

6.3 Redimensionando e Recortando

O OpenCV segue uma convenção própria para coordenadas em imagens:



Figura 7: Sistema de Coordenadas no OpenCV.

Sabendo como funciona o sistema de coordenadas do OpenCV agora podemos fazer um teste de como redimensionar uma imagem, para isso usamos a função *resize()*, caso seja necessário saber quais são as dimensões da imagem podemos usar o método *.shape*. Esse método retorna três valores, o primeiro é a altura, o segundo é a largura e o terceiro é o número de canais da imagem.

Para o redimensionamento da imagem devemos informar qual imagem será redimensionada e depois inserir uma tupla contendo as dimensões da altura e largura nessa respectiva ordem.

```
1 import cv2
2
3 imagem = cv2.imread("imagens/cg125.png")
4
5 # obtendo altura, largura e numero de canais
6 altura, largura, canais = imagem.shape
7
8 # redimensionando a imagem
9 imagem_redimensionada = cv2.resize(imagem, (int(altura/2), int(
    largura/2)))
10
11 # obtendo altura e largura da nova imagem
12 nova_altura, nova_largura, _ = imagem_redimensionada.shape
13
14 # mostrando as dimensoes de cada imagem
15 cv2.putText(imagem, f"shape: ({altura}, {largura}, {canais})", (10,
    altura - 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
16 cv2.putText(imagem_redimensionada, f"shape: ({nova_altura}, {
    nova_largura}, {canais})", (10,nova_altura - 20), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)
17
18 cv2.imshow("Imagem Original", imagem)
19 cv2.imshow("Imagem Redimensionada", imagem_redimensionada)
20
21 cv2.waitKey(0)
22
```

Listing 16: Redimensionando uma imagem.

Note que as dimensões da imagem devem ser valores inteiros, pois estamos tratando de pixels, nesse caso não admite-se valores com vírgula. No código também estamos adicionando nas imagens as dimensões de cada uma delas através da função *putText()*.



Figura 8: Redimensionando imagens com OpenCV.

Para recortar uma imagem é muito simples, não precisamos de nenhuma função específica depois que temos uma imagem já lida no OpenCV através da função *imread()* pois agora ela é uma matriz e para definirmos a região de interesse para o recorte apenas devemos indicar os intervalos de pixels que desejamos ver. Lembrando que o primeiro intervalo refere-se a altura e o segundo a largura da imagem.

```
1 import cv2
2
3 imagem = cv2.imread("imagens/cg125.png")
4
5 # obtendo altura, largura e numero de canais
6 altura, largura, canais = imagem.shape
7
8 # redimensionando a imagem
9 imagem_redimensionada = cv2.resize(imagem, (int(altura/2), int(
    largura/2)))
10
11 # obtendo altura e largura da nova imagem
12 nova_altura, nova_largura, _ = imagem_redimensionada.shape
13
14 # mostrando as dimensoes de cada imagem
15 cv2.putText(imagem, f"shape: ({altura}, {largura}, {canais})", (10,
    altura - 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
16 cv2.putText(imagem_redimensionada, f"shape: ({nova_altura}, {
    nova_largura}, {canais})", (10,nova_altura - 20), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)
17
18 # imagem recortada
19 imagem_recortada = imagem[0:altura//2, largura//2:largura]
20
21 cv2.imshow("Imagem Original", imagem)
22 cv2.imshow("Imagem Redimensionada", imagem_redimensionada)
23 cv2.imshow("Imagem Recortada", imagem_recortada)
24
25 cv2.waitKey(0)
26
```

Listing 17: Imagem recortada no OpenCV.

6.4 Formas e Textos

Para inserir formas como círculos, retângulos, linhas ou textos é bem simples, para isso já existem funções prontas no OpenCV, são elas: *circle()*, *rectangle()*, *line()* e *putText()*. Cuidado ao inserir os pontos de origem de cada forma, pois a sequência de informações de altura e largura é diferente do que é retornado pelo método *shape* já explicado anteriormente, as vezes é largura e altura nessa ordem e as vezes é altura e largura. Não é necessário decorar como cada função recebe esses parâmetros, quando você estiver testando identificará o padrão correto.

```
1 import cv2, numpy as np
2
3 # gerando uma imagem preta com numpy
4 imagem = np.zeros((512,512, 3), np.uint8)
5
6 # colorindo uma imagem completamente
7 imagem_azul = imagem.copy()
8 imagem_azul[:] = 255, 0, 0
9
10 # criando linhas
```

```

11 cv2.line(imagem, (0,0), (imagem.shape[1],imagem.shape[0]),
    (255,0,0), 5)
12
13 # criando retangulos
14 altura_retangulo, largura_retangulo = 100,100
15 centro_retangulo = (largura_retangulo//2, altura_retangulo//2)
16 centro_imagem = (imagem.shape[1]//2, imagem.shape[0]//2)
17 print((imagem.shape[1]//2), (imagem.shape[1]//2) - centro_retangulo
    [1])
18
19 cv2.rectangle(imagem, (centro_imagem[1] - centro_retangulo[1],
    centro_imagem[0] - centro_retangulo[0]), (centro_imagem[1] +
    centro_retangulo[1],centro_imagem[0] + centro_retangulo[0]),
    (125,240,0), 2)
20
21 # retangulos preenchidos
22 cv2.rectangle(imagem, (centro_imagem[1] - 20,centro_imagem[0] - 20)
    , (centro_imagem[1] + 20,centro_imagem[0] + 20), (0,0,255), cv2.
    FILLED)
23
24 # criando circulos
25 cv2.circle(imagem, (imagem.shape[1] - 30, 30), 30, (255,0,0),cv2.
    FILLED)
26 cv2.circle(imagem, (imagem.shape[1] - 30, 30), 15, (255,255,255),
    2)
27
28 # inserindo textos
29 cv2.putText(imagem, "OPENCV", (centro_imagem[0], 50), cv2.
    FONT_HERSHEY_COMPLEX, 1, (125,125,125), 1)
30
31 cv2.imshow("Imagem numpy", imagem)
32 cv2.imshow("Imagem numpy colorida", imagem_azul)
33 cv2.waitKey(0)
34

```

Listing 18: Inserindo formas e textos no OpenCV.

Resultado do código anterior:

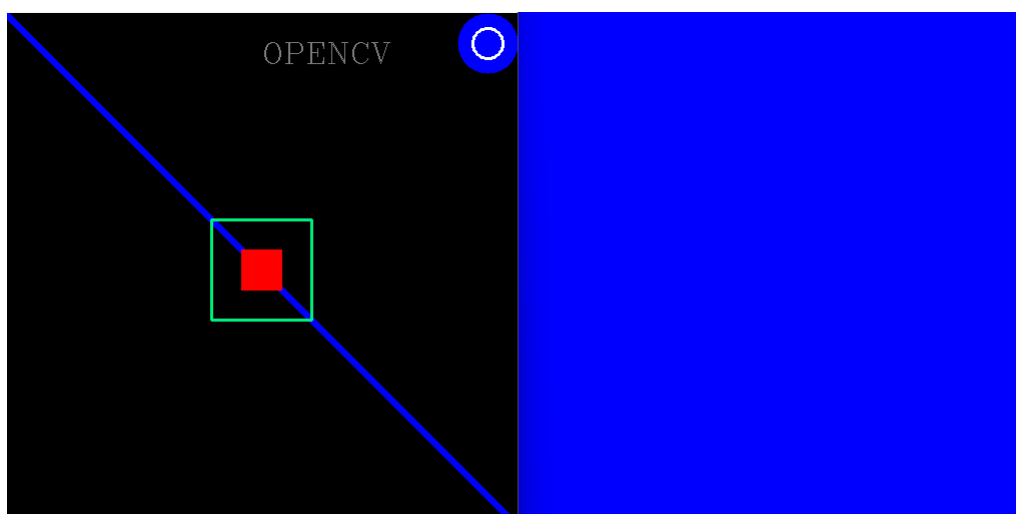


Figura 9: Formas e Textos com OpenCV.

6.5 Distorções e Perspectivas

Através do OpenCV podemos escanear imagens que possam estar em até certo ponto distorcidas ou então em perspectiva. Para isso usamos duas funções, *getPerspectiveTransform()* para calcular a matriz de transformação que pega os pontos de entrada da imagem distorcida e transforma em uma matriz conforme os pontos de referência para a saída, para isso é necessário indicar dois vetores do tipo float com o pacote numpy.

Posteriormente podemos planificar a porção desejada da imagem utilizando a função *warpPerspective* que recebe a imagem base, a matriz de transformação e as novas medidas de largura e altura para a imagem.

```
1 import cv2, numpy as np
2
3 imagem = cv2.imread("imagens\cartas.png")
4
5 # definifindo altura e largura
6 largura, altura = 250, 350
7
8 # obtendo os pontos de interesse
9 pontos1 = np.float32([[165,111], [276,92], [215,264], [344,234]])
10 pontos2 = np.float32([[0,0], [largura, 0], [0, altura], [largura,
    altura]])
11
12 # criando a matriz de transformacao
13 matriz = cv2.getPerspectiveTransform(pontos1, pontos2)
14
15 # obtendo a imagem planificada
16 imagem_planificada = cv2.warpPerspective(imagem, matriz, (largura,
    altura))
17
18 cv2.imshow("Imagem Original", imagem)
19 cv2.imshow("Imagem Planificada", imagem_planificada)
20 cv2.waitKey(0)
21
```

Listing 19: Distorções e Perspectivas com OpenCV.

O resultado obtido é o seguinte:

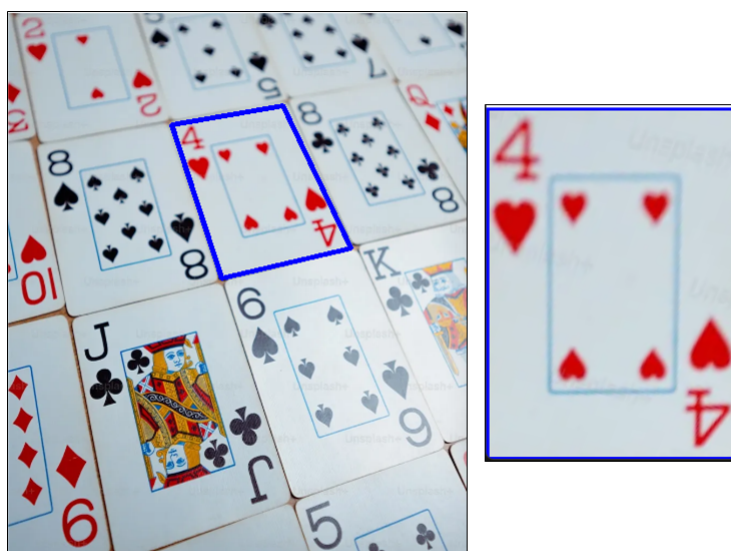


Figura 10: Formas e Textos com OpenCV.

6.6 Juntando Imagens

Para juntar imagens podemos usar algumas funções do pacote numpy pois depois de uma imagem ser lida pelo OpenCV ela é basicamente um matriz de números com o valor de cada pixel da imagem. Porém há algumas limitações, usando as funções *vstack()* e *hstack()* para juntar imagens na vertical ou na horizontal essa tarefa será apenas possível se as imagens tiverem o mesmo número de canais e dimensões. Por isso vamos usar uma função desenvolvida pessoalmente para correção desses erros, assim poderemos juntar imagens de diferentes canais e também redimensioná-las quando necessário.

```
1 import cv2, numpy as np
2
3
4 import cv2
5 import numpy as np
6
7 def empilhar_imagens(escala, matriz_imagens):
8     linhas = len(matriz_imagens)
9     colunas = len(matriz_imagens[0])
10    linhas_disponiveis = isinstance(matriz_imagens[0], list)
11    largura = matriz_imagens[0][0].shape[1]
12    altura = matriz_imagens[0][0].shape[0]
13
14    if linhas_disponiveis:
15        for x in range(0, linhas):
16            for y in range(0, colunas):
17                if matriz_imagens[x][y].shape[:2] == matriz_imagens[0][0].
18                    shape[:2]:
19                    matriz_imagens[x][y] = cv2.resize(matriz_imagens[x][y],
20                        (0, 0), None, escala, escala)
21                else:
22                    matriz_imagens[x][y] = cv2.resize(matriz_imagens[x][y], (
23                        matriz_imagens[0][0].shape[1], matriz_imagens[0][0].shape[0]),
24                        None, escala, escala)
25                if len(matriz_imagens[x][y].shape) == 2:
26                    matriz_imagens[x][y] = cv2.cvtColor(matriz_imagens[x][y],
27                        cv2.COLOR_GRAY2BGR)
28
29    imagem_branca = np.zeros((altura, largura, 3), np.uint8)
30    hor = [imagem_branca] * linhas
31    hor_con = [imagem_branca] * linhas
32
33    for x in range(0, linhas):
34        hor[x] = np.hstack(matriz_imagens[x])
35
36    ver = np.vstack(hor)
37
38    else:
39        for x in range(0, linhas):
40            if matriz_imagens[x].shape[:2] == matriz_imagens[0].shape
41                [:2]:
42                matriz_imagens[x] = cv2.resize(matriz_imagens[x], (0, 0),
43                    None, escala, escala)
44            else:
45                matriz_imagens[x] = cv2.resize(matriz_imagens[x], (
46                    matriz_imagens[0].shape[1], matriz_imagens[0].shape[0]), None,
47                    escala, escala)
48            if len(matriz_imagens[x].shape) == 2:
49                matriz_imagens[x] = cv2.cvtColor(matriz_imagens[x], cv2.
50                    COLOR_GRAY2BGR)
```

```

40
41     hor = np.hstack(matriz_imagens)
42     ver = hor
43
44     return ver
45
46
47 imagem = cv2.imread("imagens\cg125_menor.png")
48
49 # juntando duas imagens na horizontal
50 imagem_horizontal = np.hstack((imagem, imagem))
51
52 # juntando duas imagens na vertical
53 imagem_vertical = np.vstack((imagem, imagem))
54
55 # maneira mais adequada de fazer a juncao de imagens
56 imagens = empilhar_imagens(0.5, ([imagem, imagem], [imagem, imagem]
57                                  ))
58 cv2.imshow("Imagem Horizontal", imagem_horizontal)
59 cv2.imshow("Imagem Vertical", imagem_vertical)
60 cv2.imshow("Imagens", imagens)
61 cv2.waitKey(0)
62

```

Listing 20: Juntando imagens com OpenCV.

6.7 Detecção de Cores

A detecção de cores nos permite isolar áreas de interesse na imagem observada, para isso podemos seguir a seguinte abordagem:(a)converter a imagem RGB para HSV(b)criar uma ferramenta de controle deslizante para controlar os valores de Hue, Saturation e Value a fim de obtermos o resultado desejado(c)aplicar a a detecção de cores desejada na imagem alvo.

Para convertermos a imagem de RGB para HSV usamos a função *cvtColor()* agradaada ao método *.COLOR_BGR2HSV*. Depois criamos o controle deslizante a partir da função *resizeWindow()*, também é necessário criar uma janela para mostrar esse controle deslizante, usamos então a função *namedWindow()*. Cada parâmetro a ser adicionando no controle deslizante é definido pela função *createTrackbar()* e posteriormente poderá ser acessado pela função *getTrackbarPos()*.

Finalizando, então criamos uma mascara que possuirá os valores de hsv desejados e então aplicamos essa mascara na imagem inicial , usando respectivamente as funções *inRange()* e *bitwise_and()*

```

1 import cv2, numpy as np
2
3
4 def empty(value):
5     ...
6
7
8 # criando um controle deslizante
9 cv2.namedWindow("ControleDeslizante")
10 cv2.resizeWindow("ControleDeslizante", 640, 240)
11 cv2.createTrackbar("Hue Min", "ControleDeslizante", 0, 179, empty)
12 cv2.createTrackbar("Hue Max", "ControleDeslizante", 179, 179, empty
13 )
14 cv2.createTrackbar("Sat Min", "ControleDeslizante", 0, 255, empty)

```

```

14 cv2.createTrackbar("Sat Max", "ControleDeslizante", 255, 255, empty
    )
15 cv2.createTrackbar("Val Min", "ControleDeslizante", 0, 255, empty)
16 cv2.createTrackbar("Val Max", "ControleDeslizante", 255, 255, empty
    )
17
18 while True:
19     # criando um display
20     display = np.zeros((240,240,3), np.uint8)
21
22     imagem = cv2.imread("imagens\cg125_menor.png")
23
24     # alterando padrao de representacao de cores BGR para HSV
25     imagem_hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)
26
27     # obtendo os valores do hsv
28     hue_min = cv2.getTrackbarPos("Hue Min", "ControleDeslizante")
29     hue_max = cv2.getTrackbarPos("Hue Max", "ControleDeslizante")
30     sat_min = cv2.getTrackbarPos("Sat Min", "ControleDeslizante")
31     sat_max = cv2.getTrackbarPos("Sat Max", "ControleDeslizante")
32     val_min = cv2.getTrackbarPos("Val Min", "ControleDeslizante")
33     val_max = cv2.getTrackbarPos("Val Max", "ControleDeslizante")
34     menor = np.array([hue_min, sat_min, val_min])
35     maior = np.array([hue_max, sat_max, val_max])
36
37     # mostrando os valores dos controles deslizantes
38     cv2.putText(display, f"Hue_min: {hue_min}", (10,25), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
39     cv2.putText(display, f"Hue_max: {hue_max}", (10,100), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
40     cv2.putText(display, f"Sat_min: {sat_min}", (10,50), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
41     cv2.putText(display, f"Sat_max: {sat_max}", (10,125), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
42     cv2.putText(display, f"Val_min: {val_min}", (10,75), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
43     cv2.putText(display, f"Val_max: {val_max}", (10,150), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2)
44
45     mascara = cv2.inRange(imagem_hsv, menor, maior)
46     imagm_resultado = cv2.bitwise_and(imagem, imagem, mask=mascara)
47
48     cv2.imshow("CG125", imagem)
49     cv2.imshow("CG125-hsv", imagem_hsv)
50     cv2.imshow("Display", display)
51     cv2.imshow("Mascara", mascara)
52     cv2.imshow("Resultado", imagm_resultado)
53     if cv2.waitKey(1) == ord('q'):
54         break
55
56 cv2.destroyAllWindows()
57

```

Listing 21: Detectando cores específicas com OpenCV.

6.8 Contornos/Detecção de Formas