Polycarpo Souza Neto

Relatório - Vaga de Projetista de Sistemas Embarcados/*Machine Learning*

Q.1 - Utilizando OpenCV, conte a quantidade de objetos na imagem abaixo (20 escores).

Resposta.:
Para resolução dessa questão foi implementado um código em python, utilizando openCV (cv2). A metodologia adotada foi:

Na metodologia empregada para a detecção e quantificação de objetos em imagens, o processo inicia com o carregamento da imagem utilizando a função **cv2.imread** da biblioteca **OpenCV**. Este passo é essencial pois ao trabalhar com um único canal de cor, reduz-se a complexidade dos dados, facilitando a manipulação subsequente. Em seguida, aplica-se um limiar fixo de 128 para a binarização da imagem. Esse *thresholding* transforma os pixels com valores acima do limiar em branco (255) e os abaixo em preto (0), criando uma representação binária para a etapa de detecção de contornos.

A detecção dos contornos por meio da função **cv2.findContours**, que identifica as bordas dos objetos na imagem binarizada com base nas diferenças de intensidade.. Para melhorar o contraste da imagem e, consequentemente, a detecção de contornos, é realizada uma equalização de histograma na imagem original em escala de cinza. Este procedimento redistribui a luminosidade da imagem, destacando características antes não visíveis.

Após a equalização, a imagem é submetida novamente ao processo de binarização e detecção de contornos para avaliar a eficácia do pré-processamento. Os objetos detectados tanto na imagem original são então contabilizados. Cada objeto é anotado diretamente na imagem com um número sequencial, posicionado no centroide do contorno, calculado a partir dos momentos do contorno.

A seguir, temos a Figura 1, onde exibi-se a imagem original ao lado da imagem com o *count* de objetos, a saber, 115 o resultado.

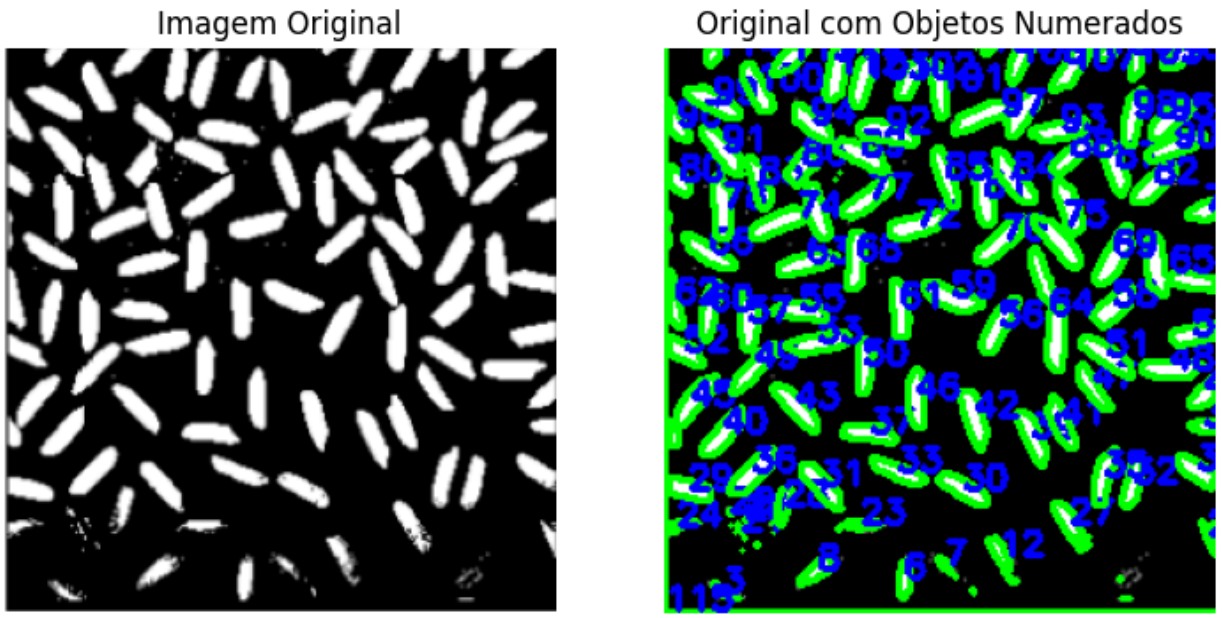

Figura 1. Resultado da contagem, Do lado esquerdo a imagem fornecida e no direito, com a identificação/count de objetos.

Q2 - Dada a imagem abaixo, conte a quantidade de pessoas (30 escores):

Resposta.:

O procedimento inicia com a verificação da disponibilidade local do arquivo de pesos 'yolov3-tiny.weights' da rede YOLOv3-tiny, que é essencial para o funcionamento do modelo de detecção. Isso assegura que o modelo de rede neural possa ser inicializado sem interrupções. Após a obtenção do arquivo de pesos, a rede é configurada e inicializada usando os parâmetros especificados no arquivo 'yolov3-tiny.cfg'.

A imagem indicada no enunciado é carregada do disco utilizando a função **cv2.imread**. Para adaptar a imagem ao input esperado pela rede neural, uma cópia é feita para cada tipo de detecção desejada—uma para a detecção de pessoas. A imagem é transformada em um 'blob' usando a função **cv2.dnn.blobFromImage**, que redimensiona a imagem para 416x416 pixels e normaliza seus valores de pixel, facilitando assim a sua compatibilidade com os requisitos de entrada da rede neural.

O modelo YOLOv3-tiny processa o 'blob' por meio de uma passagem para frente, gerando detecções que são analisadas em várias camadas. Cada detecção é avaliada por um vetor que inclui a classe do objeto detectado, o *confidence* da detecção e as coordenadas da caixa delimitadora. As detecções são filtradas para identificar especificamente pessoas (‘classe 0”), com um limiar de confiança estabelecido em 0.3 (arbitrário). As caixas delimitadoras correspondentes são ajustadas para corresponder às dimensões reais da imagem.

A técnica de Supressão de Não-Máximos (NMS) é aplicada para reduzir redundâncias nas caixas delimitadoras detectadas, mantendo apenas as mais significativas para cada pessoa identificada. Este processo envolve a comparação das sobreposições entre as caixas e a seleção daquelas com a maior confiança de detecção, eliminando caixas que representam o mesmo objeto múltiplas vezes.

O resultado encontrado usando essa “arquitetura” é exibido na Figura 2 a seguir.

Figura 2 - Identificação de pessoas com YoloV3-tiny.

Percebe-se que o resultado não foi dos mais satisfatórios, sendo identificadas apenas **03 pessoas**, então propõe-se um modelo desafiador como contraparte ao resultado exibido, só que neste caso, usando o YoloV4, onde foram identificadas **06 pessoas**. Segue na Figura 3 o resultado.

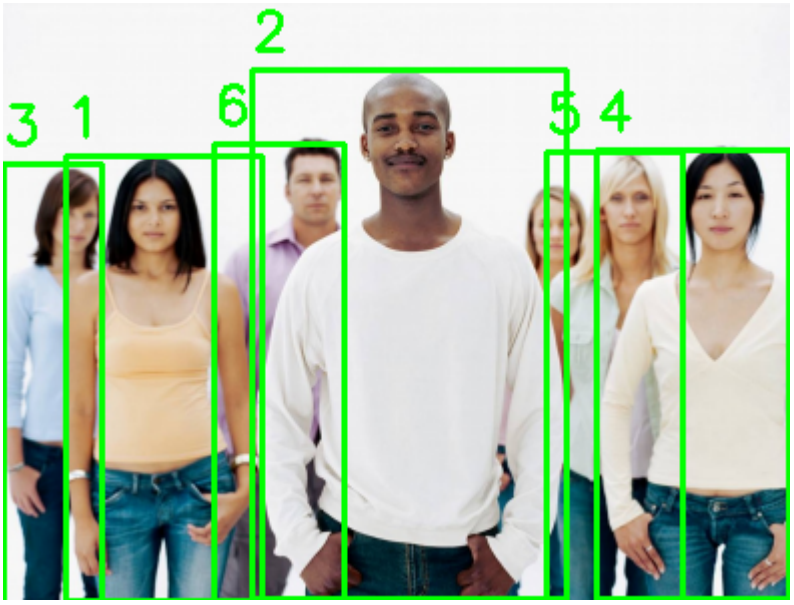

Figura 2 - Identificação de pessoas com YoloV4.

Q3 - Treine um classificador de tipos de carros utilizando DeepLearning. (50 escores)

- ☐ Utilize o banco de imagens BMW-10 dataset, disponível em: Cars Dataset (https://www.kaggle.com/datasets/jessicali9530/stanford-cars-dataset);
- ☐ O classificador deve diferenciar somente as classes 3,4 e 5 do banco BMW-10 e
- ☐ uma classe classe indefinida. Assim, o classificador deve conseguir classificar as
- ☐ classes 3, 4, 5 e indefinida;
- ☐ Separe o banco em 70 % para treinamento e 30% para teste;

Resposta.:

Inicialmente, os metadados das imagens de carros foram carregados de um arquivo .mat ('bmw10_annos.mat'), contendo informações sobre as classes e os nomes dos arquivos das imagens. O script mapeou classes específicas para consideração no modelo. Os arquivos foram divididos em conjuntos de treinamento e teste usando a função **train_test_split**, garantindo uma distribuição estratificada das classes. Observa-se pela divisão de treino e teste em 70/30%.

- *Found 358 images belonging to 4 classes.*

- *Found 154 images belonging to 4 classes.*

Foi definida uma arquitetura de CNN (rede neural convolucional(utilizando o *framework* **TensorFlow Keras**, com camadas convolucionais seguidas de camadas de *pooling* e uma camada de *flatten* para preparar os dados para as camadas densas finais. A rede incluiu **dropout** para regularização (mitigação de *overffiting)* e uma camada de saída softmax para classificação multiclasse. O modelo foi compilado com o otimizador Adam e configurado para monitorar a acurácia e a perda (*loss*) durante o treinamento.

Os dados de treinamento foram aumentados e normalizados usando geradores de imagem, **ImageDataGenerator**, facilitando o manuseio de imagens em lote e aumentando a variabilidade dos dados sem necessidade de armazenar fisicamente as imagens alteradas. O modelo foi treinado por 10 épocas, utilizando os dados de treinamento e validando com um conjunto de teste para monitorar a generalização do modelo fora da amostra de treinamento.

Mantendo o número de épocas em 10 e otimizando a nível de código os parâmetros de *dropout* e *learning_rate*, temos que a melhor acurácia possível é de 72,07% para o caso onde os parâmetros são 0,7 de dropout e 0,0005 de laerning rate.

Tabela1. Aferição de acuráciaotimizando parâmetros (treino).

learning rate	dropout	acurácia (teste)
1	0,3	0.7077922224998474
1	0,5	0.7077922224998474
1	0,7	0.7142857313156128
0,0005	0,3	0.7142857313156128
0,0005	0,5	0.701298713684082
0,0005	**0,7**	**0.7207792401313782**
0,0001	0,3	0.701298713684082
0,0001	0,5	0.701298713684082
0,0001	0,7	0.701298713684082