Disciplina:

Análise de Imagem e Visão Computacional

Professor: Octavio Santana

PUC Minas

Aula 1:

Introdução à Visão Computacional e Processamento de Imagens

Professor: Octavio Santana

PUC Minas

PUC Minas

Objetivos da Aula

Ao final desta aula, os alunos serão capazes de:

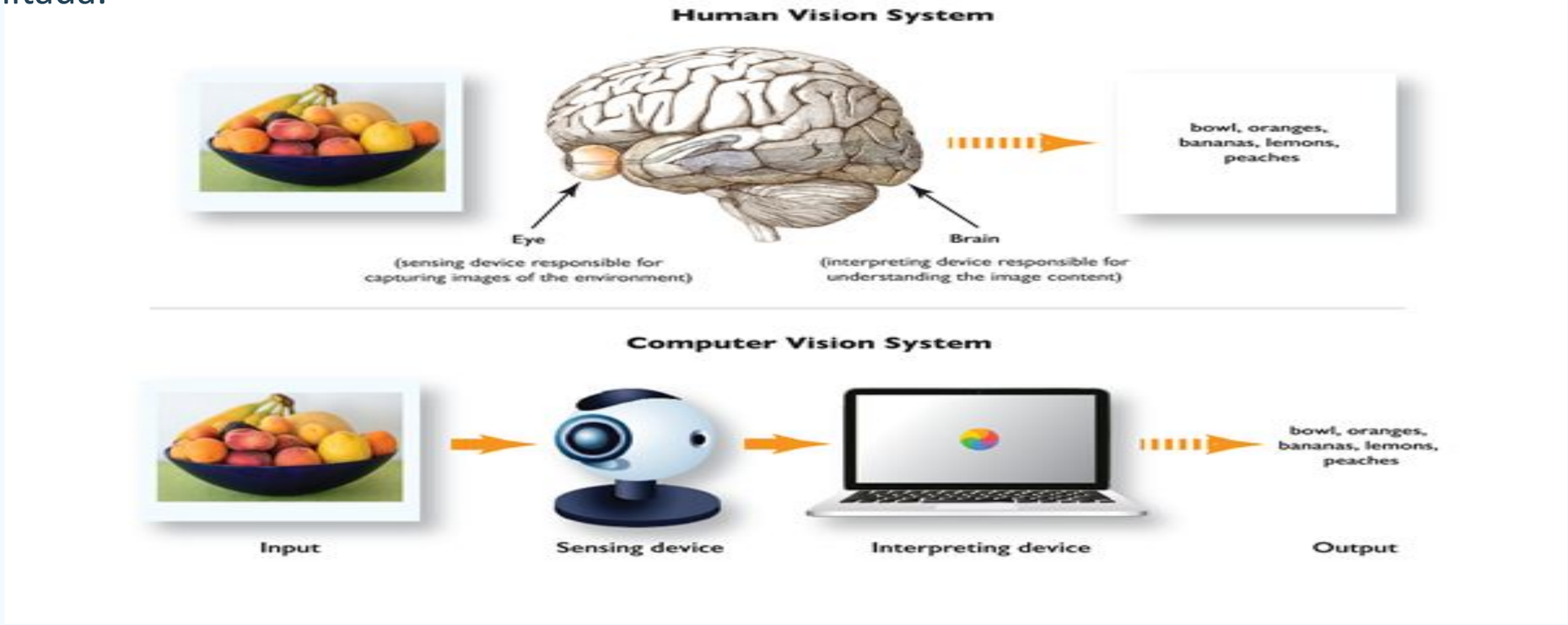
1. Compreender o conceito de Visão Computacional e suas aplicações.
2. Diferenciar os tipos de Visão Computacional.
3. Entender os modelos de representação de imagem.
4. Realizar manipulação básica de imagens utilizando OpenCV.

PUC Minas

Introdução à Visão Computacional

O que é Visão Computacional?

A visão computacional é o campo da inteligência artificial que se concentra em replicar partes da complexidade do sistema de visão humana e permitir que os computadores identifiquem e processem objetos em imagens e vídeos da mesma forma que os humanos. Até recentemente, a visão computacional funcionava apenas em capacidade limitada.

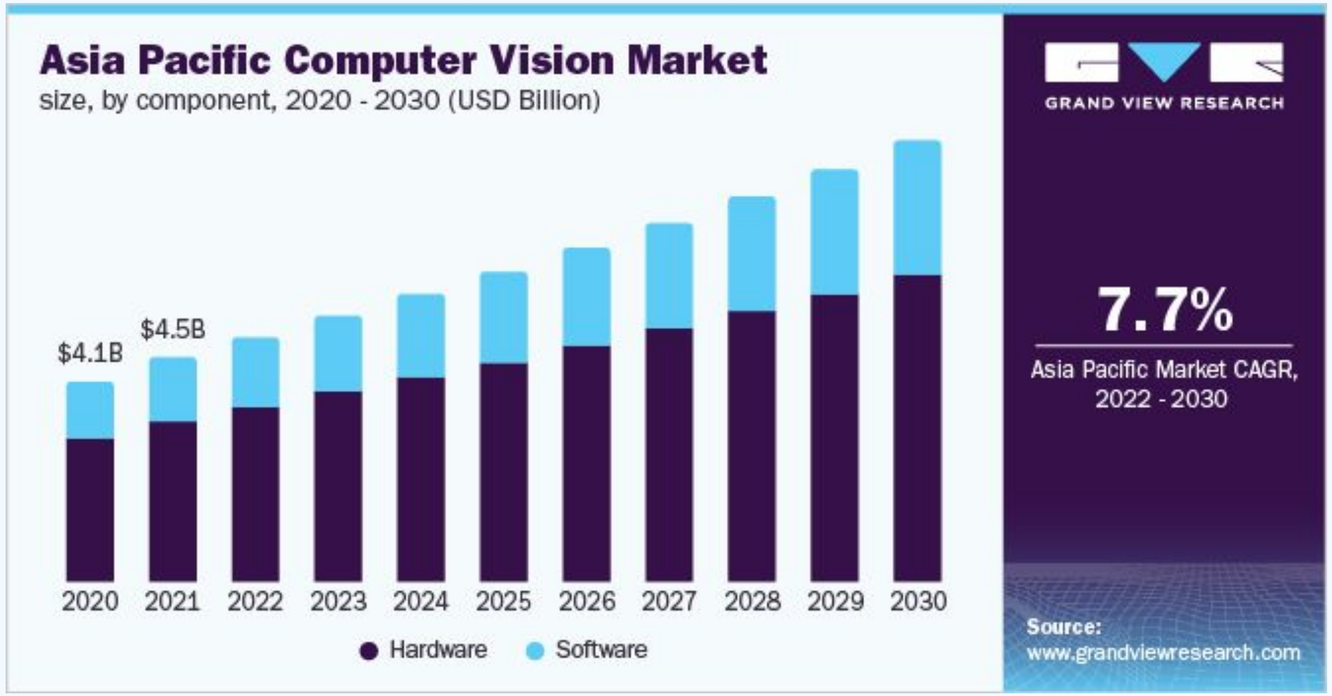


PUC Minas

Contextualização

Os primeiros experimentos em visão computacional começaram na década de 1950 e foram usados comercialmente pela primeira vez para distinguir entre texto digitado e manuscrito na década de 1970. Hoje, os aplicativos para visão computacional cresce exponencialmente.

”O mercado global de Visão Computacional foi avaliado em US$11,22 bi em 2021 e **deve crescer a uma taxa de 7% de 2022 a 2030.**”



Vantagens do uso da Visão Computacional

- ➔ Suas aplicações estão bem cercadas para a gestão de qualidade total, acesso a áreas perigosas e alto processamento.
- ➔ Processos de alta velocidade que exigem uma olhar mais “atento” e que acompanhe os movimentos, prezando qualidade e segurança.

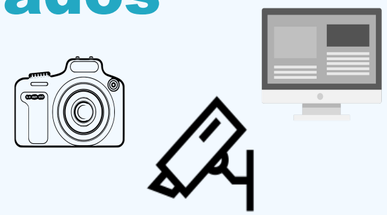
Suas aplicações são extensas e vamos pontuar algumas:

- Reconhecimento facial
- Reconhecimento de objetos
- Reconhecimento de placas de veículos
- Inspeção de falhas em linhas de produção

PUC Minas

As iniciativas de Visão Computacional seguem o seguinte fluxo

Coleta dos Dados



As fontes podem ser câmeras, páginas da internet, fotografias, etc

Armazenamento

Os dados coletados são catalogados e armazenados. Geralmente na nuvem.

Aprendizagem

Modelos de IA são criados para responder perguntas pertinentes ao negócio.

Decisão

As saídas dos modelos são utilizadas para auxiliar nas tomadas de decisão.

PUC Minas

Custo do uso de Visão Computacional

Soluções de Visão Computacional são, em sua maioria, mais caras do que as demais de Ciência de Dados.

Antes de armazenar as imagens são tratadas para obedecer um padrão tanto de qualidade quanto de tamanho.

Para exemplificar, suponhamos que uma imagem custe R$0,10 no armazenamento e R$1,00 no aprendizado.

Uma câmera normalmente gera 32 imagens por segundo, o que resulta em 2.664.800 imagens por dia.

Se fossemos salvar e processar todas, daria R$3.041.280,00 por dia.

Custo do uso de Visão Computacional

Algumas ações já são adotadas para mitigar o custo de armazenamento:

- Manter armazenada uma quantidade menor de imagens por período de tempo. Exemplo: uma imagem a cada cinco minutos.
- Manter histórico mais recente armazenado, assim como as soluções de câmeras funcionam. Exemplo: 30 dias de armazenamento.
- Armazenar apenas os metadados das soluções e não as imagens em si.
- Selecionar o que é gravado por meio da solução implementada.

Custo do uso de Visão Computacional

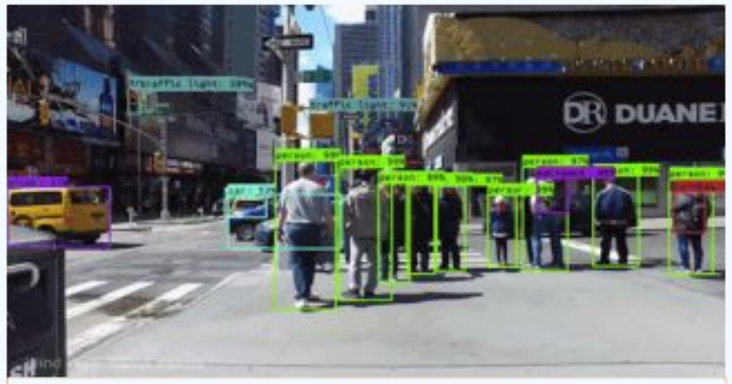
Diminuir o custo de aprendizado é uma tarefa difícil, pois:

- Visão computacional demanda computadores mais poderosos.
- Assim como o negócio é mutável, as soluções de ciência de dados também. Uma mudança de cenário demanda atualização da solução. O que consiste, por muitas vezes, em maiores quantidades de imagens para processar.
- A quantidade de imagens necessárias para implementar uma solução depende de diversos fatores, como a quantidade de fontes de dados, a quantidade de itens para serem identificados, a pluralidade de cenários, etc.

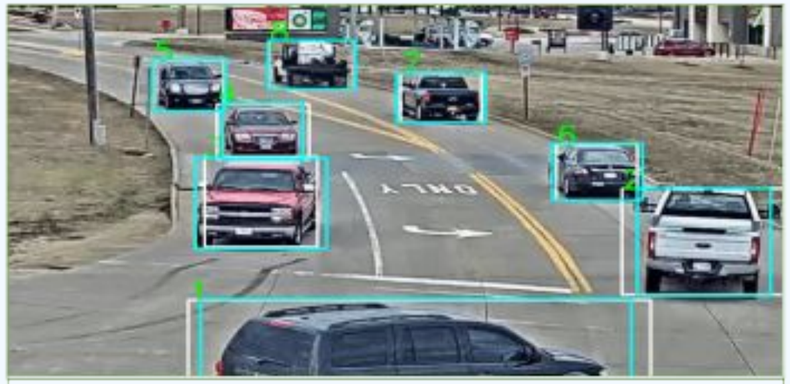
Principais Aplicações

PUC Minas

Algumas técnicas de Visão Computacional



Detecção/Classificação



Rastreamento

Segmentação

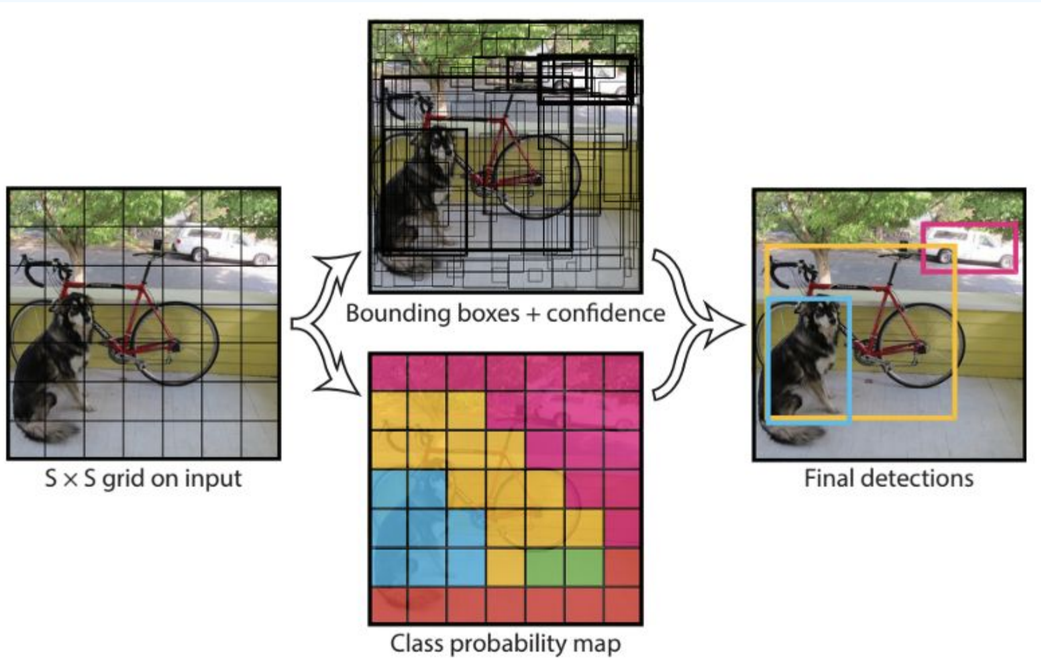
Geração

Pose Estimation

Modelos de Detecção e Classificação de Objetos

Alguns modelos de detecção e classificação de objetos:

- R-CNN
- SSD
- YOLO



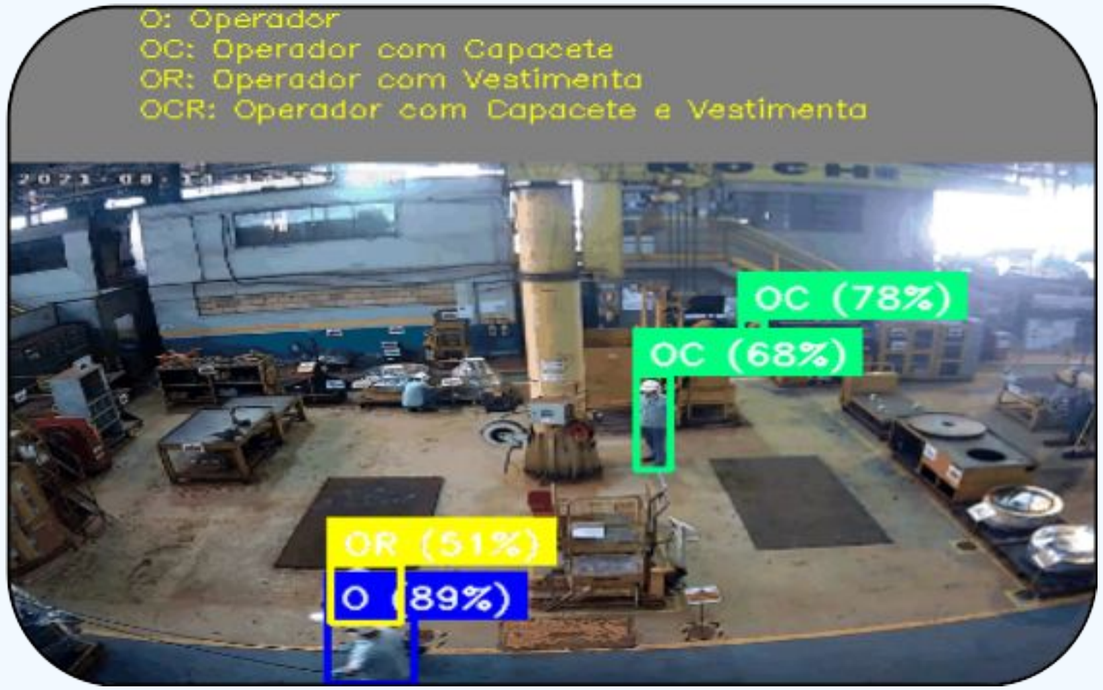
Em YOLO uma única rede convolucional prediz tanto os bounding boxes quanto às probabilidades de pertencer a classe de cada objeto detectado. Para isso, YOLO funciona da seguinte forma:

- toma-se uma imagem e divide-se-a em um grid SxS de células;
- usando o grid como referência, gera-se m bounding boxes;
- bounding boxes com probabilidade acima de um limiar são selecionados e usados para localizar o objeto dentro da imagem.

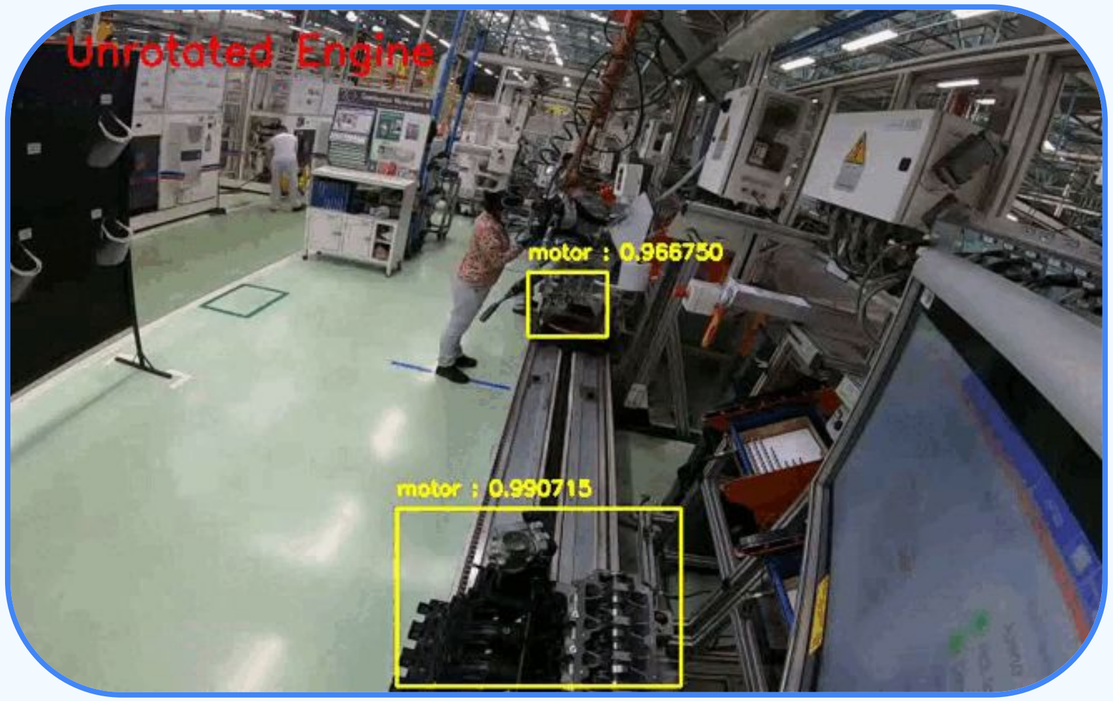
PUC Minas

Modelos de Detecção e Classificação de Objetos

Verificação de Uso de EPIs



Checklist de processos e procedimentos



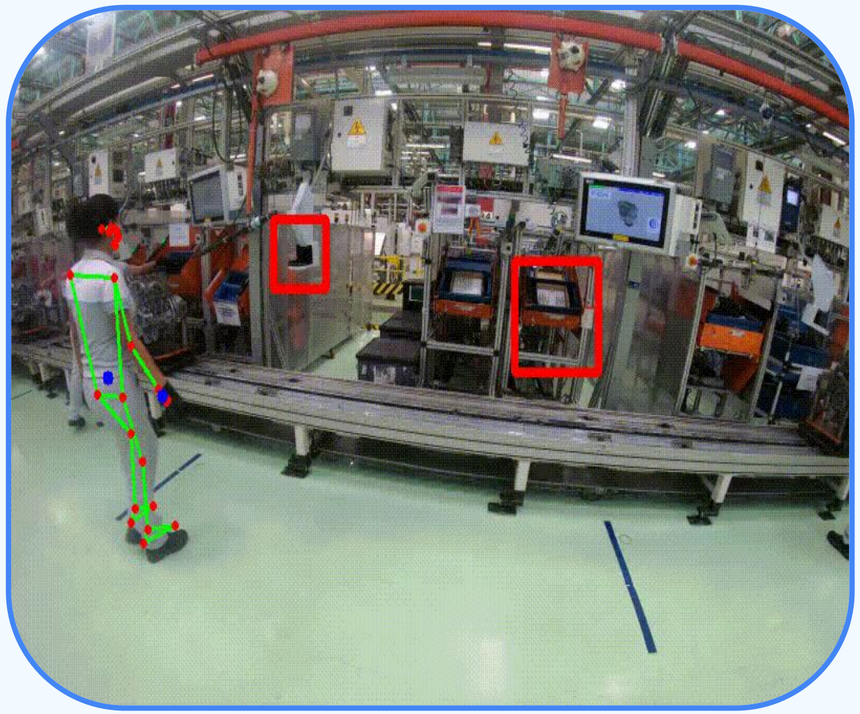
PUC Minas

Pose Estimation

Basicamente esse modelo detecta vários pontos referente às articulações e características das pessoas, como ombros, joelhos, mãos e etc. Esses dados podem ser usados para direcionar exibições interativas, jogos, experiências em realidade virtual, etc.

Alguns modelos:

- OpenPose
- PoseNet
- AlphaPose



Tipos de Visão Computacional

Tipos de Visão Computacional

- Visão Computacional Tradicional
- Visão Computacional Baseada em Aprendizado de Maquina
- Visão Computacional Baseada em Aprendizado Profundo

Visão Computacional Tradicional

Utiliza técnicas clássicas de processamento de imagens, como:

- Filtragem espacial: Aplicada para suavização de imagens, realçar bordas e remover ruídos.
- Detecção de Bordas: Usada para identificar contornos em uma imagem.
- Extração de características: Identifica padrões como cantos e texturas.

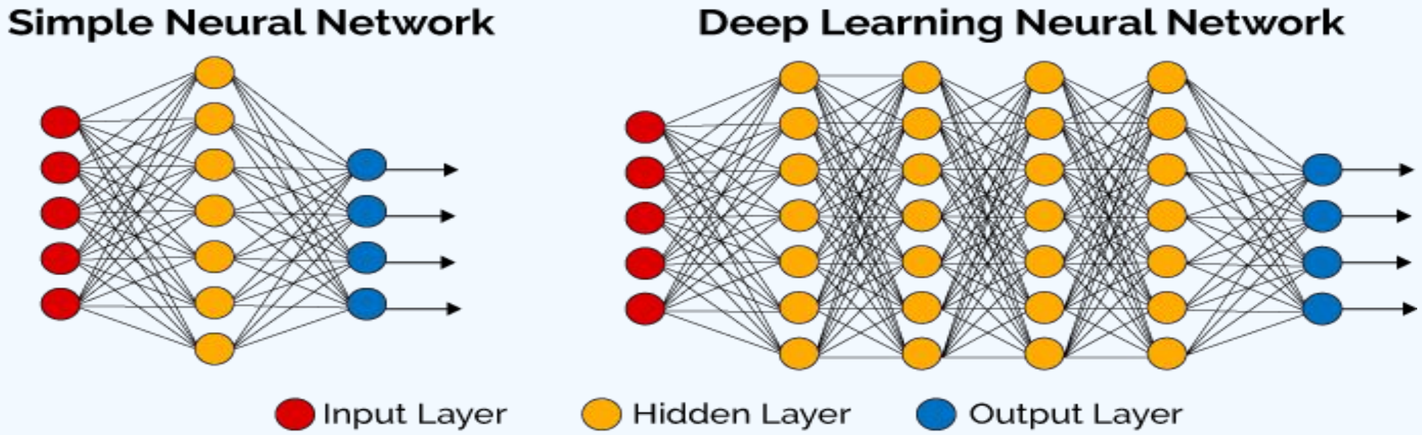
Visão Computacional baseada em Aprendizado de Máquina

Utiliza técnicas de machine learning para classificação e segmentar imagens. Algumas das técnicas mais comuns incluem:

- SVM (Support Vector Machine): Usado para classificação de objetos.
- KNN (K-Nearest Neighbords): Algoritmo baseado em proximidade de dados.

Visão Computacional com Aprendizado Profundo

Usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação.



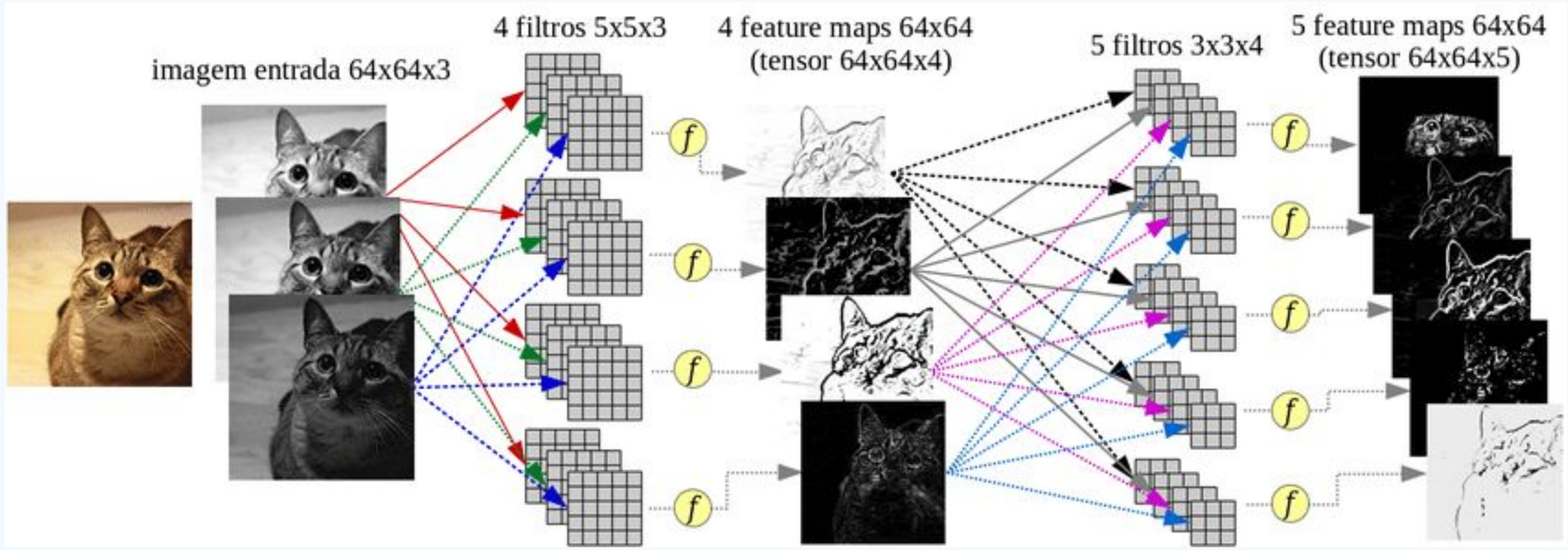
É pouco eficiente usar redes com camadas totalmente conectadas para classificar imagens. A razão é que tal arquitetura de rede não leva em conta a estrutura espacial das imagens. Por exemplo, ela trata os pixels de entrada que estão distantes e próximos exatamente no mesmo nível. Tais conceitos de estrutura espacial devem ser inferidos dos dados de treinamento.

PUC Minas

Visão Computacional com Aprendizado Profundo

Deep Learning revolucionou a Visão Computacional com redes neurais convolucionais (CNNs), permitindo modelos mais precisos para reconhecimento de padrões complexos.

É uma classe de redes neurais de aprendizado profundo. Resumindo, pense na CNN como um algoritmo de aprendizado de máquina que pode receber uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos/objetos na imagem e ser capaz de diferenciar um do outro.



PUC Minas

Introdução ao OpenCV e Manipulação de Imagens

O que é OpenCV?

OpenCV (Open Source Computer Vision Library) é uma biblioteca open-source amplamente utilizada para processamento de imagens, visão computacional e aprendizado de máquina. Desenvolvida em C++, também possui interfaces para Python e Java.

Principais recursos:

- Leitura, gravação e exibição de imagens.
- Transformações (redimensionamento, rotação, filtros).
- Detecção de bordas, rostos e objetos.
- Processamento em tempo real com câmeras.

Importando a biblioteca do OpenCV

```
import cv2
print("OpenCV version:", cv2.__version__)
```

✓ 0.0s

```
OpenCV version: 4.11.0
```

Criando uma virtualenv com nome `curso_cv`

```
python -m venv curso_cv
```

Ativando a virtualenv

```
# Windows
curso_cv\Scripts\activate

# Linux/MacOS
source curso_cv/bin/activate
```

Instalar o OpenCV

```
pip install opencv-python
```

Formulação de Imagens

Uma imagem pode ser descrita como uma função 2D, f(x,y), onde (x,y) são as coordenadas espaciais e o valor de f em qualquer ponto, (x,y), é proporcional aos níveis de brilho ou cinza da imagem. Além disso, quando ambos os valores (x,y) e de brilho de f são todos quantidades discretas finitas, a imagem é chamada de imagem digital.

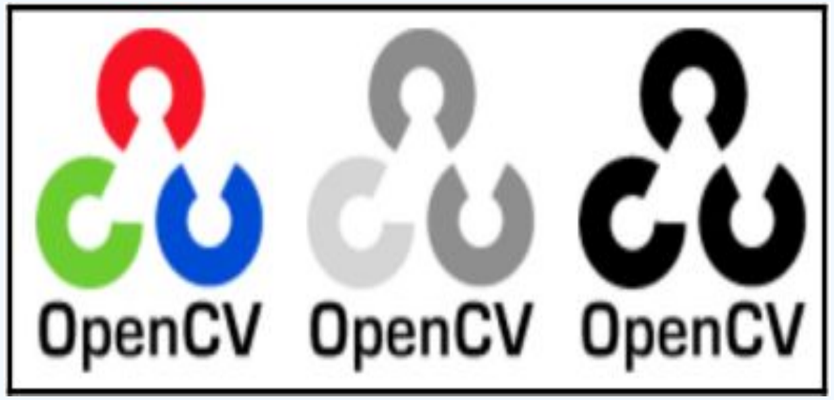
Portanto, f(x,y) assume os seguintes valores:

- $x \in [0, h-1]$, onde h é a altura da imagem
- $y \in [0, w-1]$, onde w é a largura da imagem
- $f(x,y) \in [0, L-1]$, onde L = 256 (para uma imagem de 8 bits)

Imagens em tons de cinzas são representadas por uma única função f(x, y), com valores entre 0 e 255

Imagens coloridas são representadas por três funções fR(x,y), fG(x,y) e fB(x,y) (Red, Green e Blue). Cada função segue a mesma lógica da função f(x,y) de imagens em tons de cinza.

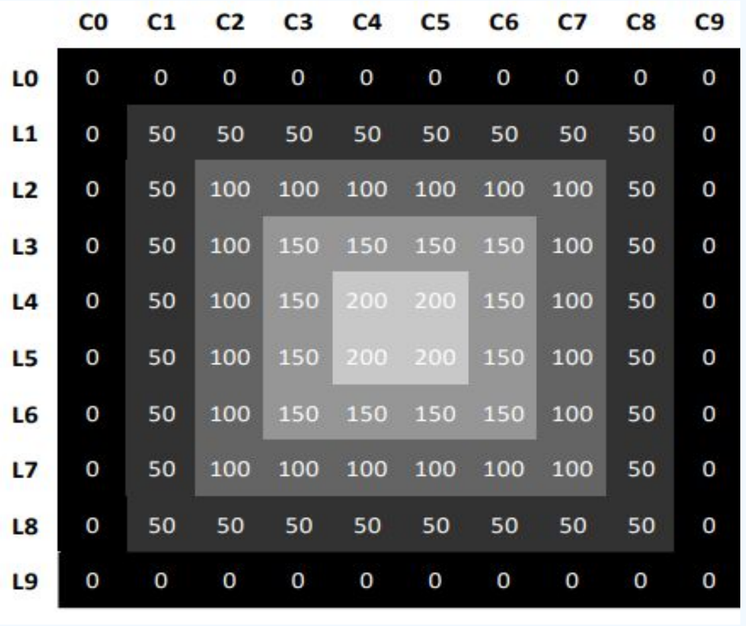
Images preto e branco (binárias) são representadas por uma única função f(x, y), que pode assumir apenas dois valores 0 (preto) e 255 (branco).



PUC Minas

Sistema de coordenadas e manipulação de pixels

- Sistema de coordenadas de pixels é baseado em (linha, coluna)
- Origem da imagem corresponde ao pixel no canto superior esquerdo
- Eixo horizontal (colunas): cresce da esquerda para a direita
- Eixo vertical (linhas): cresce da cima para baixo
- Exemplo de uma imagem 300x200 pixels:
  - Largura: 300 pixels (colunas 0 a 299)
  - Altura: 200 pixels (linhas: 0 a 199)



	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
L0	0	0	0	0	0	0	0	0	0	0
L1	0	50	50	50	50	50	50	50	50	0
L2	0	50	100	100	100	100	100	100	50	0
L3	0	50	100	150	150	150	150	100	50	0
L4	0	50	100	150	200	200	150	100	50	0
L5	0	50	100	150	200	200	150	100	50	0
L6	0	50	100	150	150	150	150	100	50	0
L7	0	50	100	100	100	100	100	100	50	0
L8	0	50	50	50	50	50	50	50	50	0
L9	0	0	0	0	0	0	0	0	0	0

Qual a valor do Pixel (2, 8)?

A linha índice 2 que é a terceira linha e na coluna índice 8 que é a nona linha possui ao valor 50.

Carregando e exibindo imagens no OpenCV

Para carregar uma imagem, utilizamos a função *cv2.imread()*, que pode receber o caminho do arquivo ou simplesmente o nome da imagem, caso esteja no diretório de trabalho:

```
imagem = cv2.imread("caminho/da/imagem.jpg", flags)
```

flags: Controla como a imagem é carregada:

- cv2.IMREAD_COLOR (padrão): Carrega em BGR (3 canais).
- cv2.IMREAD_GRAYSCALE: Carrega direto em escala de cinza (1 canal).
- cv2.IMREAD_UNCHANGED: Preserva canais (ex.: PNG com transparência).

Para exibir a imagem carregada, utilizamos *cv2.imshow()*, que cria uma janela para visualização:

```
cv2.imshow("Titulo da Janela", imagem)
cv2.waitKey(0)  # Espera até que uma tecla seja pressionada
cv2.destroyAllWindows()  # Fecha todas as janelas
```

Salvando imagens no OpenCV

Para salvar uma imagem em um arquivo, utilizamos a função *cv2.imwrite()*, que recebe o caminho/nome do arquivo de destino e a imagem a ser salva. Essa função é essencial para armazenar resultados de processamento ou imagens modificadas.

```
sucesso = cv2.imwrite("caminho/salvar/imagem_saida.jpg", imagem)
if sucesso:
    print("Imagem salva com sucesso!")
else:
    print("Erro ao salvar a imagem.")
```

Parâmetros:

- caminho/do/arquivo.jpg (string):
  - Caminho onde a imagem será salva, incluindo o nome do arquivo e extensão (ex.: "resultado.png").
  - Formatos suportados: JPEG, PNG, BMP, TIFF, etc.
- imagem (array numpy):
  - Imagem a ser salva (em formato BGR, RGB ou escala de cinza).

Retorno:

- sucesso:
  - True se a imagem foi salva com sucesso.
  - False se houve erro (ex.: caminho inválido ou permissão negada).

Obs.:

A função *cv2.imwrite()* espera imagens no formato BGR (padrão OpenCV), não RGB. Se você tentar salvar uma imagem que foi convertida para RGB, a imagem será salva sem erro, mas as cores ficarão invertidas (vermelho ↔ azul) porque o OpenCV assume que a matriz de pixels está em BGR.

Desenhando e escrevendo em imagens com OpenCV

Em projetos de Visão Computacional, frequentemente precisamos:

- Destacar resultados (como objetos detectados)
- Anotar informações (coordenadas, classes, confiança)
- Criar visualizações intuitivas para análise ou demonstração

O OpenCV oferece funções intuitivas para adicionar elementos gráficos diretamente em matrizes de pixels, mantendo alta performance.

Funções básicas de desenho

- Linha (*cv2.line*)

```
cv2.line(
    img,            # Imagem de destino (numpy array)
    (x1, y1),       # Ponto inicial (x, y)
    (x2, y2),       # Ponto final (x, y)
    (B, G, R),      # Cor (Blue, Green, Red)
    espessura,      # Espessura da linha (px)
    lineType=cv2.LINE_AA  # Tipo: LINE_AA (anti-aliasing) ou LINE_8 (padrão)
)
```

Exemplo:

```
cv2.line(imagem, (10, 10), (200, 200), (0, 255, 0), 3, cv2.LINE_AA)
```

Desenhando e escrevendo em imagens com OpenCV

Funções básicas de desenho

- Retângulo (*cv2.rectangle*)

```
cv2.rectangle(
    img,            # Imagem de destino
    (x1, y1),       # Canto superior esquerdo
    (x2, y2),       # Canto inferior direito
    (B, G, R),      # Cor
    espessura,      # Espessura ou -1 (preenchimento)
    lineType        # Opcional: Tipo de linha
)
```

Exemplo:

```
cv2.rectangle(imagem, (50, 50), (150, 150), (255, 0, 0), -1)  # Preenchido
```

PUC Minas

Desenhando e escrevendo em imagens com OpenCV

Funções básicas de desenho

- Circulo (*cv2.circle*)

```
cv2.circle(
    img,              # Imagem de destino
    (x, y),           # Centro do circulo
    raio,            # Raio em pixels
    (B, G, R),        # Cor
    espessura         # -1 para preenchimento
)
```

Exemplo:

```
cv2.circle(imagem, (100, 100), 30, (0, 0, 255), 2)  # Borda vermelha
```

Desenhando e escrevendo em imagens com OpenCV

Funções básicas de desenho

- Texto (*cv2.putText*)

```
cv2.putText(
    img,            # Imagem de destino
    "Texto",        # String a ser escrita
    (x, y),         # Canto inferior esquerdo do texto
    fonte,          # Estilo (ex: cv2.FONT_HERSHEY_SIMPLEX)
    escala,         # Tamanho do texto
    (B, G, R),      # Cor
    espessura,      # Espessura da fonte
    lineType        # Opcional: Tipo de linha (ex: cv2.LINE_AA)
)
```

Fontes disponíveis:

- cv2.FONT_HERSHEY_SIMPLEX (normal)
- cv2.FONT_HERSHEY_PLAIN (fino)
- cv2.FONT_HERSHEY_SCRIPT_COMPLEX (cursivo)

Exemplo:

```
cv2.putText(imagem, "OpenCV", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (255, 255, 255), 2)
```

Carregando e Salvando Vídeos com OpenCV

Vídeos são essenciais para sistemas de visão computacional, pois representam sequências temporais de imagens (frames). O OpenCV oferece ferramentas poderosas para:

- *Processamento em tempo real:* Análise de fluxo de vídeo imediato (ex.: vigilância, reconhecimento facial).
- *Manipulação de arquivos:* Edição, conversão e extração de frames de vídeos pré-gravados.
- *Automação:* Criação de pipelines para processamento em batch (ex.: filtros, redimensionamento).

Aplicações Práticas:

- Segurança:
  - Detecção de movimento ou objetos em câmeras de vigilância.
- Entretenimento:
  - Efeitos especiais em vídeos (filtros AR, substituição de fundo).
- Pesquisa:
  - Análise de comportamento humano ou animal em estudos científicos.
- Industria:
  - Controle de qualidade em linhas de produção por visão computacional.

Carregando e Salvando Vídeos com OpenCV

No Opencv o vídeo é carregado usando a função *cv2.VideoCapture()*

```
video = cv2.VideoCapture("caminho/do/video.mp4")  # Arquivo
# ou
video = cv2.VideoCapture(0)  # Webcam (0 = câmera padrão)
```

Parâmetros:

- caminho/do/video.mp4: Vídeo pré-gravado.
- 0 (ou 1, 2, ...): Índice da câmera.

Verificando Abertura:

```
if not video.isOpened():
    print("Erro ao carregar o video!")
```

Carregando e Salvando Vídeos com OpenCV

A leitura e exibição do vídeo é feita em loop de leitura (frame a frame)

```python
cap = cv2.VideoCapture('video_01.mp4')

# Recupera os frames por segundo (FPS) do vídeo original.
fps = cap.get(cv2.CAP_PROP_FPS)
if fps <= 0:
    fps = 30

# Converte FPS para milissegundos de delay entre frames:
# Esse delay será usado no waitKey() para manter a velocidade original do vídeo
delay = int(1000 / fps)

while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break

    cv2.imshow('WebCam', frame)

    # waitKey(delay) pausa a execução pelo tempo calculado, mantendo o FPS original:
    ## Retorna -1 se nenhuma tecla for pressionada
    ## Retorna o código ASCII se uma tecla for pressionada
    ## Comparação com ord('q') permite sair ao pressionar 'q'
    if cv2.waitKey(delay) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Carregando e Salvando Vídeos com OpenCV

No Opencv o vídeo é salvo usando a função *cv2.VideoWriter()*

```
codec = cv2.VideoWriter_fourcc(*'XVID')  # Codec (ex: 'MJPG', 'MP4V')
saida = cv2.VideoWriter(
    "saida.avi",           # Nome do arquivo
    codec,                 # Codec de video
    30.0,                  # FPS (quadros por segundo)
    (640, 480)             # Resolução (largura, altura)
)
```

Gravando os frames

```
while True:
    ret, frame = video.read()
    if not ret:
        break

    saida.write(frame)  # Grava o frame no arquivo
```

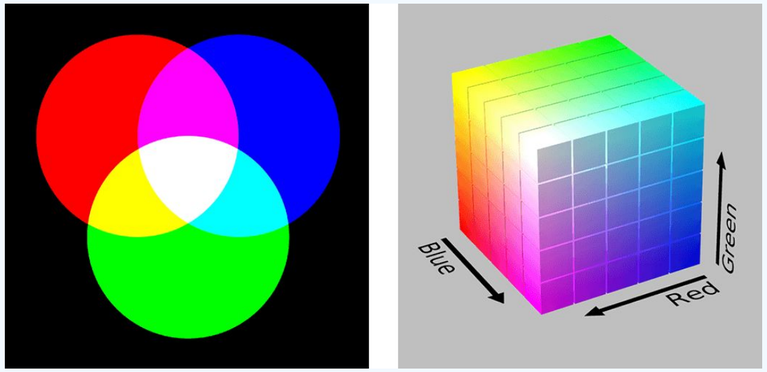
Fechando recursos

```
saida.release()
video.release()
```

PUC Minas

Representação de Imagens em Computadores

- Imagens são armazenadas como matrizes:
  - Preto e branco (P&B): matriz 2D (único canal, valores 0-255)
  - Coloridas (RGB): matriz 3D (3 canais - Red, Green, Blue)
- Cada pixel em P&B é um número inteiro (0 = preto e 255 = branco). Em imagens coloridas, cada pixel é uma tupla (R, G, B) de 8 bits sem sinal (que em Python é definido por “uint8).



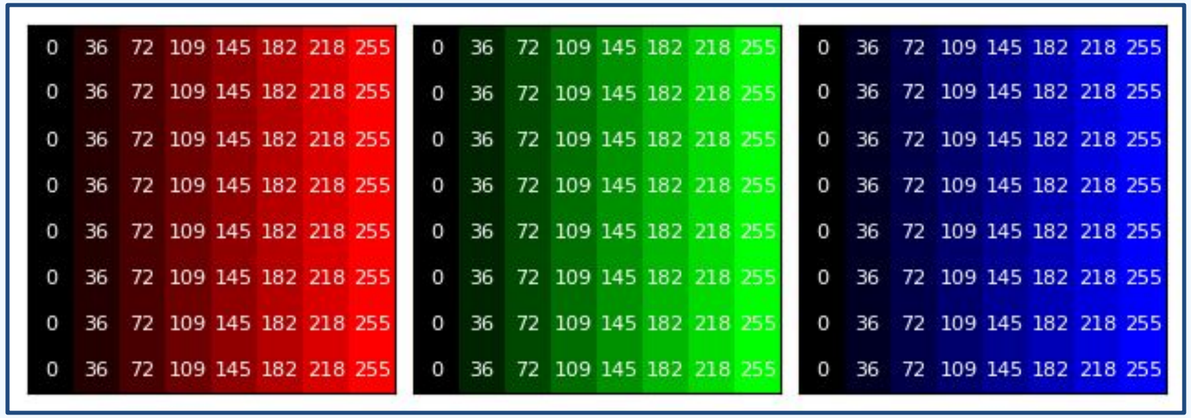
Exemplos de cores no modelo RGB:

- Preto: (0, 0, 0)
- Branco: (255, 255, 255)
- Verde: (0, 255, 0)
- Amarelo: (255, 255, 0)

➔ A imagem tem 3 matrizes que compõe o sistema RGB.

➔ Cada pixel da imagem, portanto, é composto por 3 componentes de 8 bits cada, sem sinal, o que gera 256 combinações por cor.

➔ A junção das 3 matrizes produz uma imagem colorida com capacidade de 256^3 = 16,7 milhões de cores.



Conversão de cores no OpenCV (BGR, RGB, Escala de Cinza)

O OpenCV trabalha nativamente com BGR, não RGB (diferente da maioria das bibliotecas). Os desenvolvedores iniciais do OpenCV escolheram o formato de cor BGR (em vez do RGB) porque, na época, o formato de cor BGR era muito popular entre os provedores de software e fabricantes de câmeras.

Conversões são necessárias para:

- Visualização correta (Matplotlib/outras ferramentas esperam RGB).
- Pré-processamento para algoritmos (ex.: escala de cinza para detecção de bordas).
- Isolar características específicas (ex.: canal HSV para segmentação por cor).

Quem faz a conversão?

- A função *cv2.cvtColor()* realiza o cálculo matemático dos valores dos pixels para transformar o espaço de cores.

Conversão de cores no OpenCV (BGR, RGB, Escala de Cinza)

Espaços de Cores no OpenCV

- BGR (Padrão OpenCV)
  - Ordem inversa de canais: Blue (Azul), Green (Verde), Red (Vermelho).
  - Problema comum: Se exibir uma imagem BGR como RGB, as cores ficarão distorcidas.

```
imagem_bgr = cv2.imread("imagem.jpg")  # Carrega em BGR por padrão
```

- RGB (Padrão Universal)
  - Ordem correta para exibição: Red, Green, Blue.

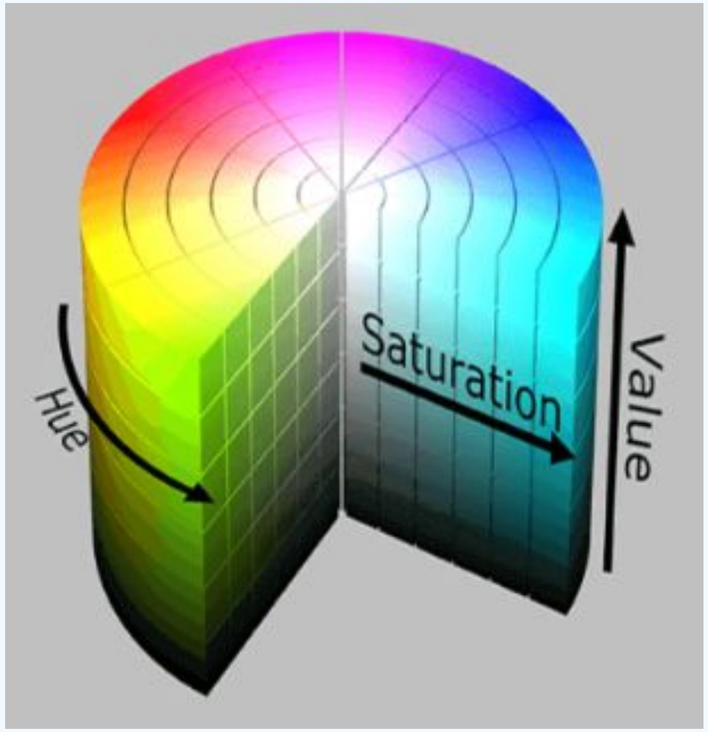
```
imagem_rgb = cv2.cvtColor(imagem_bgr, cv2.COLOR_BGR2RGB)
```

- Escala de Cinza (Grayscale)
  - Converte a imagem para um único canal (intensidade luminosa).
  - Fórmula: 0.299*R + 0.587*G + 0.114*B (pesos baseados na percepção humana).

```
imagem_cinza = cv2.cvtColor(imagem_bgr, cv2.COLOR_BGR2GRAY)
```

Cores no espaço HSV

- Amplamente utilizado em Visão Computacional e Processamento de Imagens.
- Principais vantagens:
  - Separa a cor e iluminação.
  - Facilita a segmentação de objetos coloridos.
- Diferença em relação ao RGB:
  - RGB usa canais de cores primárias
  - HSV utiliza três componentes:
    - H (Hue/Matiz): Representa a cor em círculo de 0° a 360° na teoria das cores. No OpenCV, esse intervalo é escalado para 0 a 180 (metade da escala para otimização computacional).
    - S (Saturation / Saturação): Mede a intensidade da cor, variando de 0 (tons acinzentados) a 255 (cor mais intensa).
    - V (Value/Valor): Indica o brilho da cor, onde 0 é preto e 255 é o mais claro possível.
- Muito aplicado em algoritmos de segmentação de cores.



Cores no espaço HSV

Matiz

Este é o componente que nos permite diferenciar visualmente o azul do vermelho e do verde. Representa a tonalidade da cor, o que nos permite descrever uma cor "pura" como semelhante ou diferente de outra, sem adição de preto ou branco. Pode ser vista fisicamente como o comprimento de onda dominante da cor.

-80% de matiz

imagem original

+80% de matiz

Cores no espaço HSV

Saturação

É a pureza ou intensidade da cor. Quanto maior o valor da saturação, mais pura será a cor; quanto menor, mais próxima ao seu tom de cinza ela será representada. Ao reduzir completamente a saturação de uma imagem, ela se transformará em uma imagem em tons de cinza.

-80% de saturação **imagem original** **+80% de saturação**

Cores no espaço HSV

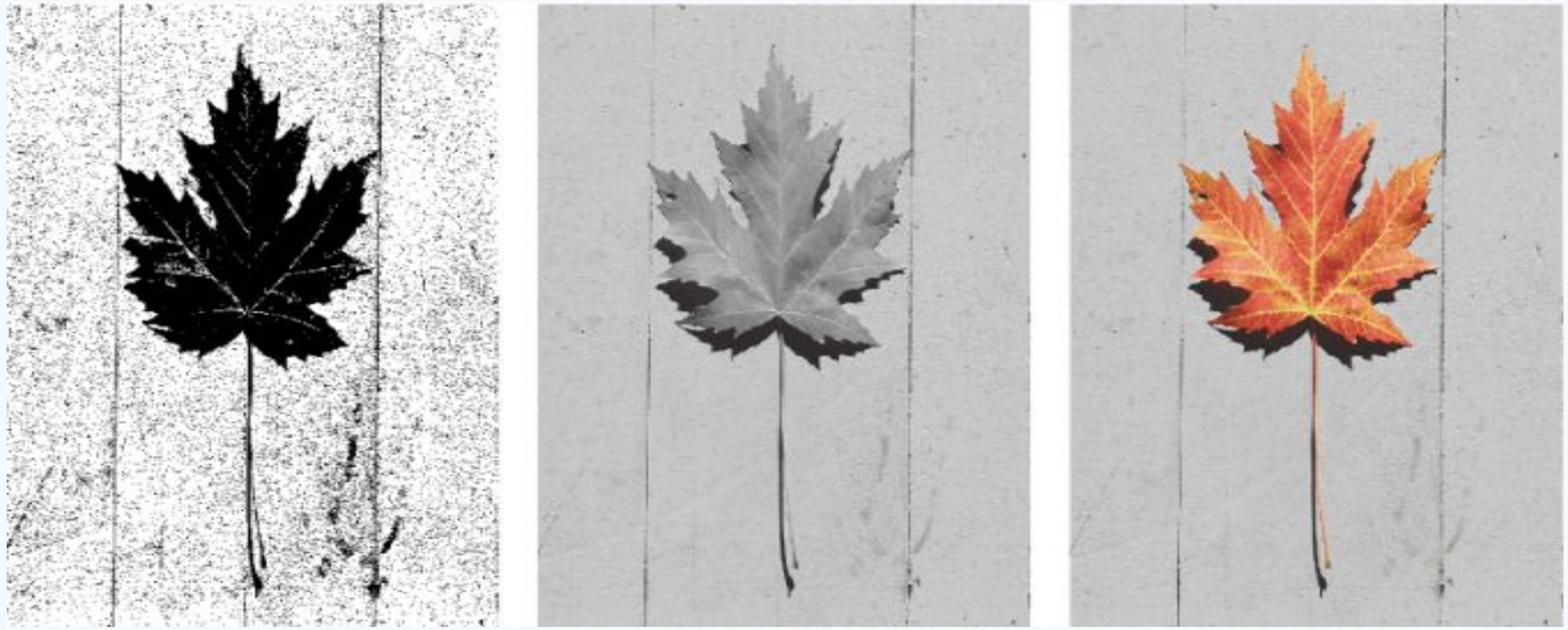
Valor

Este valor é referente ao brilho da cor, à luminosidade ou escala de claridade. Quanto maior a luminosidade, ou o valor deste componente, mais próximo ao branco a cor será representada; quanto menor, mais próxima ao preto. Ao aumentar completamente a luminosidade de uma imagem, ela se transformará em uma imagem completamente branca, mas, ao realizarmos a operação inversa, ela se transformará em uma imagem completamente preta.

-80% de luminosidade **imagem original** **+80% de luminosidade**

Histograma de cores

O histograma de cores de uma imagem é a distribuição de frequência dos níveis de cinza em relação ao número de amostras. Essa distribuição nos fornece informações sobre a qualidade da imagem, principalmente no que diz respeito à intensidade luminosa e ao contraste.



Histograma em uma imagem binária

Por possuir pixels representados apenas pela cor preta ou branca, o **histograma de cores de uma imagem binária** pode ser facilmente obtido. O total de pixels pretos ou bancos pode ser obtido percorrendo toda a matriz que representa a imagem, contando-os individualmente.

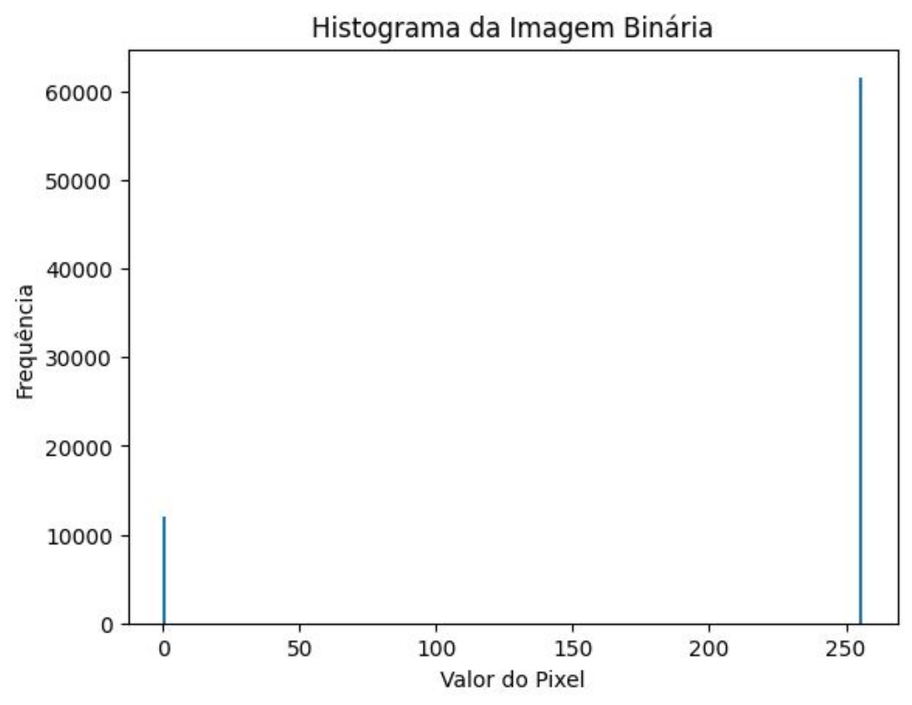
```
shape = img_bin.shape

total_preto = 0
total_branco = 0

for i in range(shape[0]):
    for j in range(shape[1]):
        if img_bin[i][j] == 0:
            total_preto += 1
        else:
            total_branco += 1

print('Total de pixels preto:', total_preto)
print('Total de pixels branco:', total_branco)
```

Observe no código que a imagem binária carregada foi salva em um arquivo do tipo Bitmap, justamente para garantir a fidelidade das cores dos pixels, sendo eles pretos ou brancos. Salvar uma imagem binária utilizando um formato compactado, como JPEG, poderia resultar em pixels com valores intermediários, em tons de cinza, não apenas pretos e brancos.

Histograma em uma imagem binária

Mesmo sendo um algoritmo simples e de fácil entendimento, esse procedimento não é muito utilizado para obter o total de pixels de cada cor. Isso porque a biblioteca Matplotlib possui a função hist, que, além de abstrair essa tarefa, gera a figura que ilustra o histograma de cores da imagem.

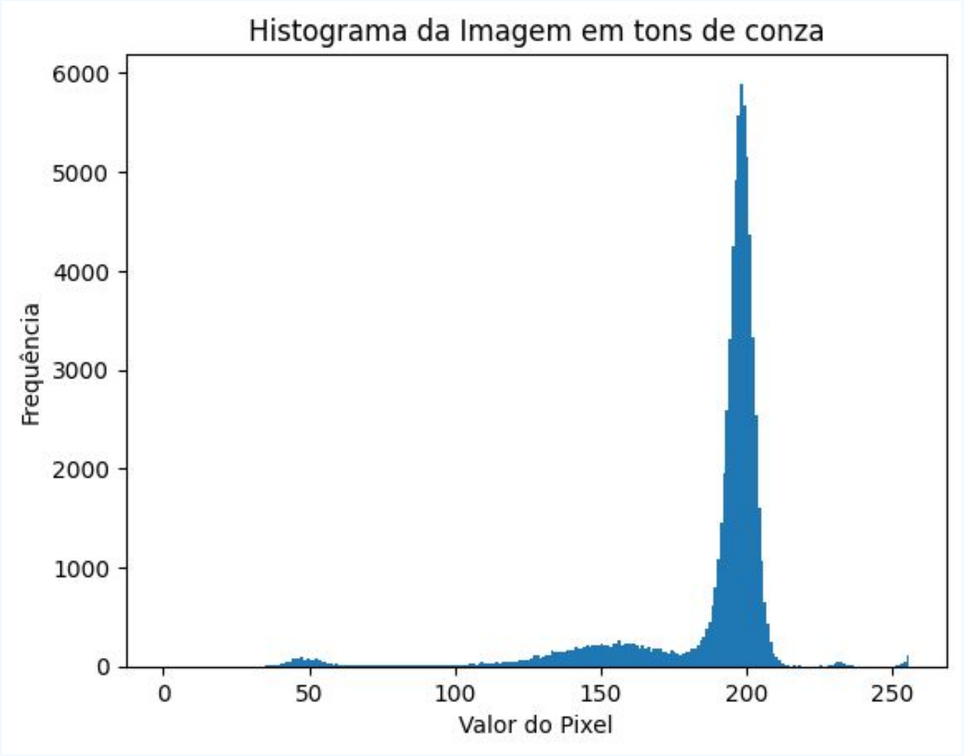
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_bin = cv2.imread('imagem_folha_binaria.bmp', cv2.IMREAD_GRAYSCALE)

plt.hist(img_bin.ravel(), bins=256, range=(0,256))
plt.xlabel('Valor do Pixel')
plt.ylabel('Frequência')
plt.title('Histograma da Imagem Binária')
plt.show()
```



O método *ravel*, aplicado no primeiro parâmetro, tem a finalidade de transformar a imagem em um vetor, isto é, organizar todos os elementos em uma estrutura, contendo uma única linha e N colunas. Em que N representa o total de pixels da imagem.

Histograma em uma imagem em tons de cinza

Nas imagens em tons de cinza, por haver grande diversidade de tons, o histograma pode ser representado em classes que contabilizam os valores em determinados intervalos. Quanto maior o número de classes, mais informações sobre a imagem podem ser representadas.

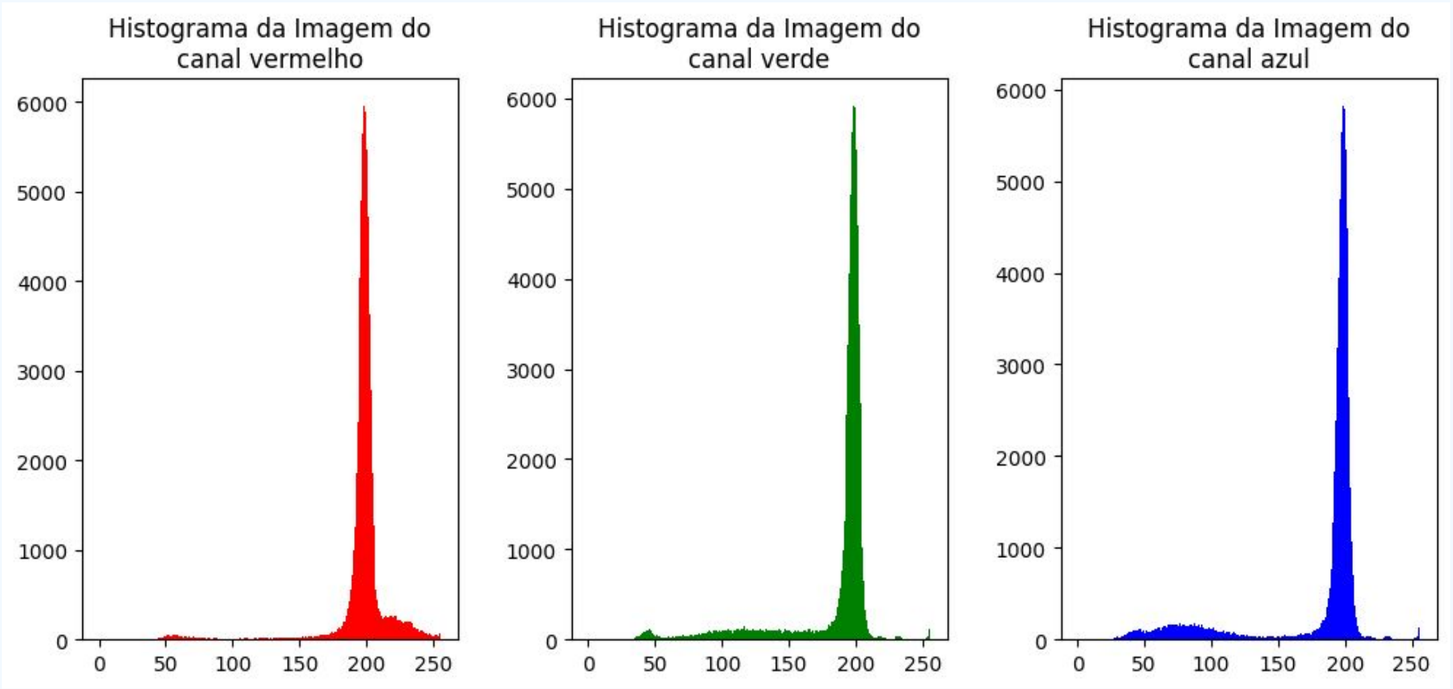


O histograma ilustrado na figura revela que a imagem processada é uma com baixo nível de contraste. Imagens com baixo nível contraste são caracterizadas por apresentarem histogramas estreitos, ou seja, muitos elementos concentrados em intervalos pequenos.

Histograma em uma imagem colorida

Uma imagem colorida possui um histograma para cada canal de cor. Uma vez que cada canal é representado por pixels em tons de cinza, o mesmo método estudado nos tópicos anteriores pode ser utilizado para plotar esses histogramas.

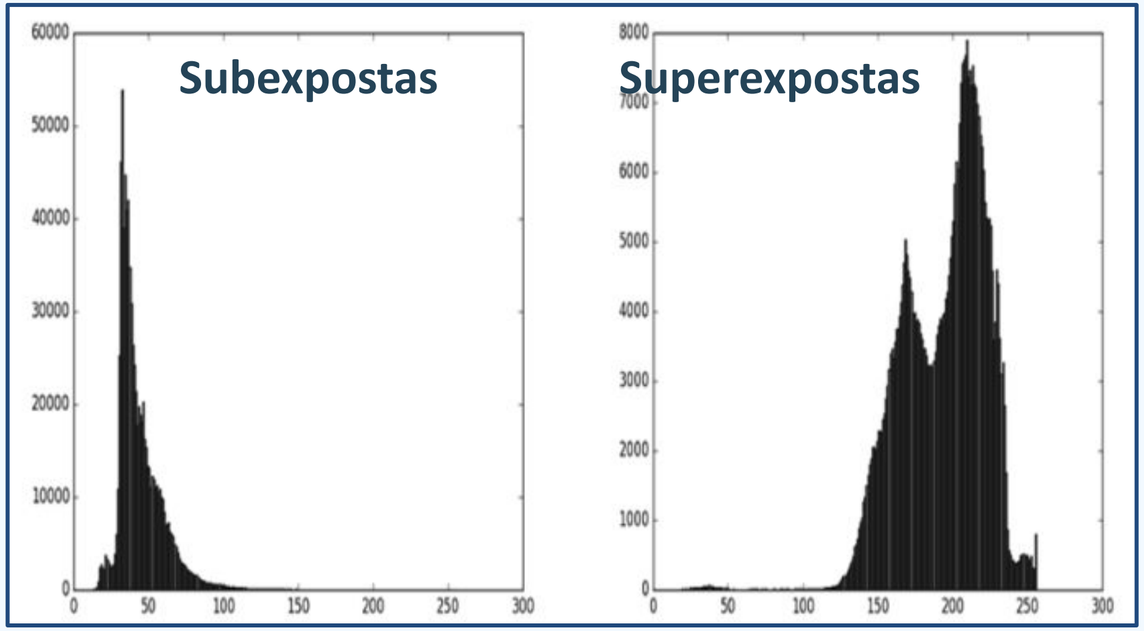
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('imagem_folha.jpg')
img_b, img_g, img_r = cv2.split(img)

plt.figure(figsize=(12,5))

plt.subplot(1,3,1)
plt.hist(img_r.ravel(), bins=256, range=(0,256), color='red')
plt.title('Histograma da Imagem do\ncanal vermelho')

plt.subplot(1,3,2)
plt.hist(img_g.ravel(), bins=256, range=(0,256), color='green')
plt.title('Histograma da Imagem do\ncanal verde')

plt.subplot(1,3,3)
plt.hist(img_b.ravel(), bins=256, range=(0,256), color='blue')
plt.title('Histograma da Imagem do\ncanal azul')

plt.subplots_adjust(wspace=0.3, hspace=0.1)

plt.show()
```

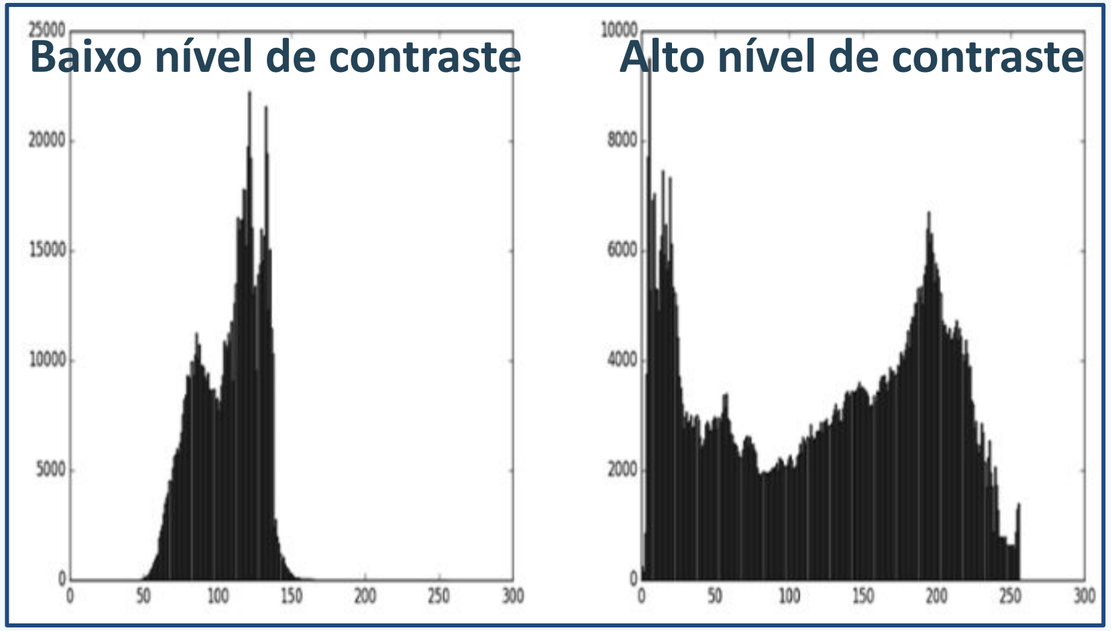


PUC Minas

Equalização de Histograma

- O histograma de uma imagem fornece informações sobre sua exposição à luz e seu nível de contraste.
- Imagens superexpostas apresentam histogramas deslocados para a direita.
- Imagens subexpostas apresentam histogramas deslocados para a esquerda.
- O contraste da imagem pode ser identificado por meio da distribuição dos valores no histograma:
  - Baixo contraste: histograma estreito, concentrado em um intervalo pequeno.
  - Alto contraste: histograma amplo, distribuído por toda a faixa de tons.





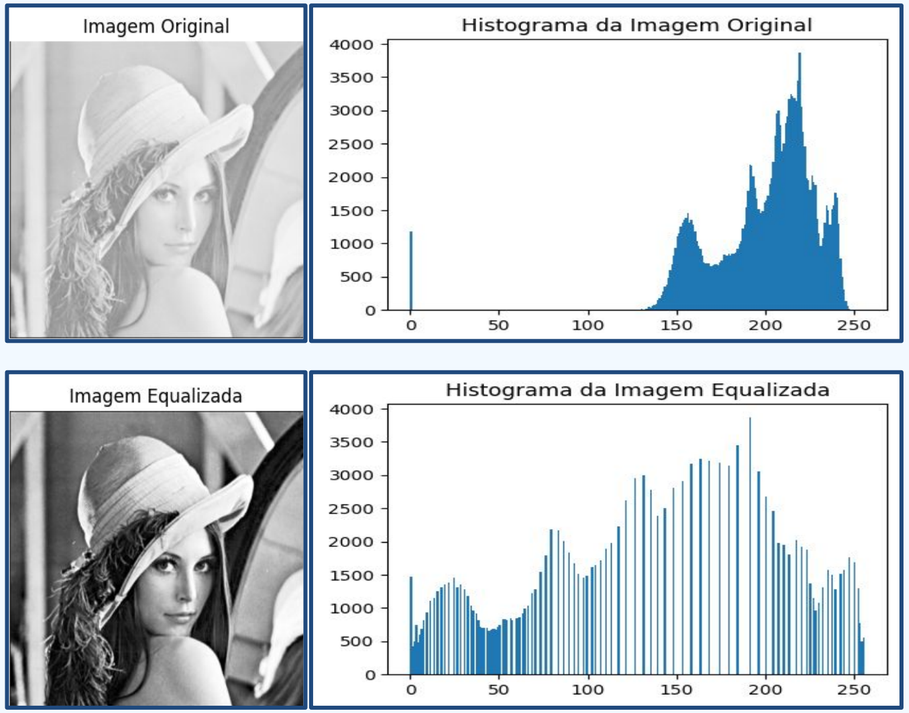
Equalização de Histograma

Importância da equalização de histograma

- Melhorar a nitidez de objetos de interesse em imagens com baixo contraste.
- Redistribuir os níveis de intensidade para ocupar toda a faixa disponível (0 a 255 em tons de cinza).
- Aplicável a imagens capturadas em condições adversas, como ambientes enfumaçados ou com iluminação inadequada

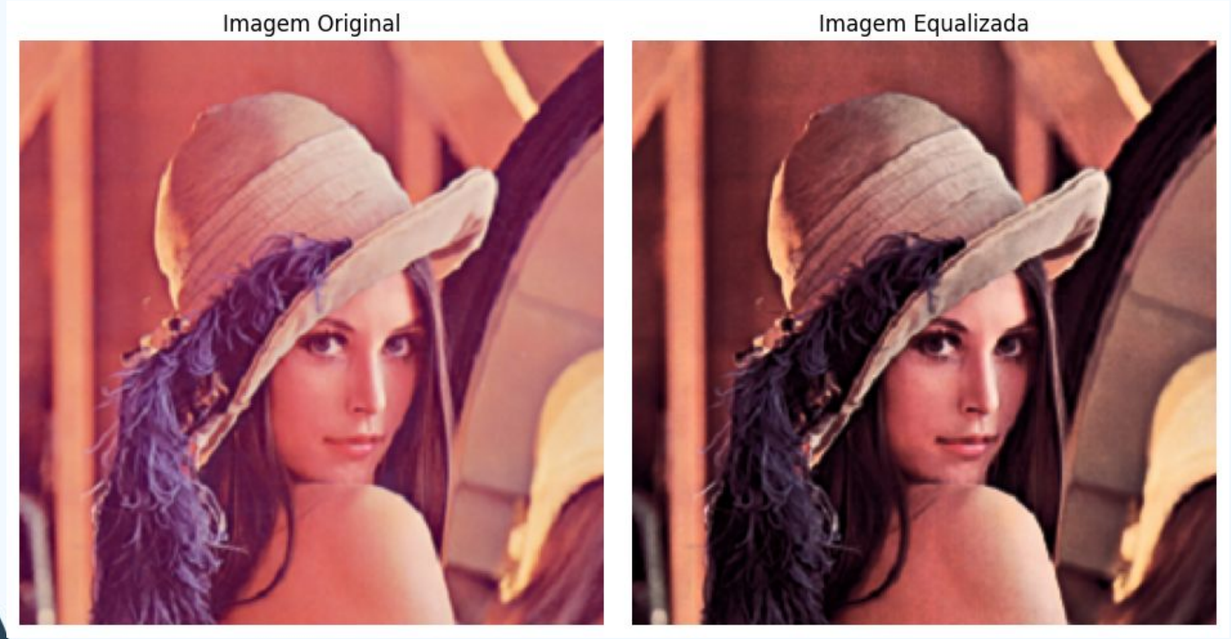
A função *cv2.equalizeHist* da biblioteca OpenCV nos permite equalizar histogramas de imagens. Ela possui como único parâmetro obrigatório a matriz que representa a imagem carregada. Executando essa função, uma nova imagem com o histograma equalizado será retornada.

```
img = cv2.imread("lena_cinza.png", cv2.IMREAD_GRAYSCALE)
img_equalizada = cv2.equalizeHist(img)
```



Equalização em imagens coloridas

- A equalização individual dos canais R, G e B pode distorcer as cores da imagem.
- A solução é converter a imagem para o espaço de cores HSV:
  - Converter a imagem de RGB para HSV.
  - Aplicar a equalização ao canal V (intensidade luminosa).
  - Converter a imagem de volta para RGB.




```python
img = cv2.imread("lenna_250.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Converter para HSV
img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

# Equalizar apenas o canal de intensidade V
img_hsv[:,:,2] = cv2.equalizeHist(img_hsv[:,:,2])

# Converter de volta para RGB
img_equalizada = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2RGB)

plt.figure(figsize=(12,8))

plt.subplot(1,2,1)
plt.imshow(img)
plt.title('Imagem Original')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(img_equalizada)
plt.title('Imagem Equalizada')
plt.axis('off')

plt.subplots_adjust(wspace=0.05, hspace=0.3)
plt.show()
```

PUC Minas

Conceitos de resolução

O que é Resolução Espacial?

- A resolução refere-se à quantidade de pixels em uma imagem.
- Expressa em largura x altura (exemplo: 1920x1080 pixels).
- Quanto maior a resolução, maior a quantidade de detalhes visíveis na imagem.

Diferença entre Pixel e Dimensão Física

- Pixel: Unidade menor de uma imagem digital, representando um ponto de cor.
- Dimensão Física: Mede o tamanho real da imagem em unidades como polegadas ou centímetros.
- Pixels ≠ Dimensão Física: O tamanho físico da imagem depende da densidade de pixels por polegada (PPI), não da contagem de pixels

Mesma fotografia representada em diferentes resoluções

Conceitos de resolução

Como calcular o PPI?

- O PPI pode ser calculado utilizando a seguinte fórmula:
  - PPI = largura (pixels) / largura da imagem (polegadas)
  - PPI = altura (pixels) / altura da imagem (polegadas)
- Exemplo:
  - Digamos que temos uma imagem com 4 polegadas (10,16cm) de largura x 4 polegadas (10,16cm) e que ela possua 1200 pixels de largura x 1200 pixels de altura. Então ela terá 300 pixels por polegada ou seja 300 PPI

Converter pixels para centímetros

- **Centímetros = Pixels ÷ PPI × 2,54**
- Do exemplo anterior, 1200 pixels com densidade de 300 PPI equivalem a 10,16 centímetros.

Tipos de resolução comuns

- HD (1280x720): Resolução básica com boa qualidade para transmissões e vídeos online
- Full HD (1920x1080): Padrão popular para telas e conteúdo digital.
- 4K (3840x260): Alta definição com grande quantidade de detalhes.

PUC Minas

Conceitos de resolução

Resolução de Bit ou Resolução de Intensidade

- Determina a quantidade de valores de intensidade ou cor que um pixel pode representar.
- Por exemplo:
  - Imagem binária (apenas cores preto e branco) é representado com 1 bit por pixel (0=preto e 1 = branco)
  - Imagem em tons de cinza representado por 8 bits por pixel pode conter até 256 tons diferentes.
  - Imagem colorida representadas em 3 canais distintos (RGB) onde cada canal tem 8 bits, totalizando 24 bits por pixel.

Observe a diferença entre as duas fotografias. A imagem à esquerda possui pixels representados em até 256 tons de cinza, diferente da imagem à direita, com pixels representados em até 4 tons de cinza.

Resolução temporal

- Esta está empregada ao sistema de captura contínua de imagens, como os vídeos, representando o número de imagens capturadas em um dado intervalo de tempo. Geralmente, usa-se a unidade frames por segundo (fps) para definir a resolução temporal de um vídeo, em que cada frame é representado por uma única imagem.

Extensões de arquivos de imagem e compressão

As imagens manipuladas no OpenCV podem ser vistas como matrizes retangulares de pixels RGB, mas isso não significa que elas sejam armazenadas, transmitidas ou processadas diretamente nesse formato. Para otimizar armazenamento e transmissão, diferentes formatos utilizam compressão com ou sem perdas.

Principais formatos suportados pelo OpenCV

O OpenCV trabalha com diversos formatos de imagem, incluindo:

✅ **Bitmap (BMP, DIB)** – Formato de imagem raster sem compressão, podendo armazenar diferentes profundidades de cor e perfis de cor.

✅ **JPEG / JPG / JPE** – Formato raster que usa compressão com perdas (lossy) para reduzir o tamanho do arquivo, sendo amplamente utilizado para fotografias digitais.

✅ **JPEG 2000 (JP2)** – Variante do JPEG que usa compressão baseada em wavelets, permitindo melhor qualidade e menos artefatos em compressões mais agressivas.

✅ **PNG** – Formato raster com compressão sem perdas (lossless), criado como alternativa ao GIF. Mantém alta qualidade sem perder informações.

Extensões de arquivos de imagem e compressão

Principais formatos suportados pelo OpenCV

O OpenCV trabalha com diversos formatos de imagem, incluindo:

✅ **PNM (PBM, PGM, PPM)** – Formatos simples e portáteis para troca de imagens. Podem armazenar imagens binárias (PBM), tons de cinza (PGM) ou coloridas (PPM).

✅ **TIFF (TIF)** – Formato versátil que pode conter imagens com ou sem compressão, sendo muito utilizado em aplicações profissionais.

Compressão: Lossless vs Lossy

📌 **Lossless (Sem perdas)** – Nenhuma informação da imagem original é perdida. O arquivo pode ser restaurado à sua forma original. Exemplos: PNG, BMP, TIFF (sem compressão).

📌 **Lossy (Com perdas)** – Parte dos detalhes da imagem é removida para reduzir o tamanho do arquivo. Muito usado para otimizar espaço e facilitar transmissões. Exemplos: JPEG, JPEG 2000.

Conclusão

Objetivos Alcançados

1. Compreendemos o conceito de Visão Computacional e suas diversas aplicações no mundo real.
2. Exploramos os diferentes tipos de Visão Computacional e suas particularidades.
3. Aprendemos sobre os modelos de representação de imagens.
4. Realizamos manipulação básica de imagens utilizando a biblioteca OpenCV.