Filtragem Espacial

Prof. Yanky Fonte Boa

Índice

- Introdução: Contextualização do tema
- Fundamentos de Processamento de Imagens
- O que é Filtragem Espacial?
- Matemática da Filtragem Espacial
- Tipos de Filtros
- Codificação - Exemplo com Python
- Desafios e Dicas

Introdução

- As imagens estão presentes em todos os aspectos de nossa vida cotidiana, desde redes sociais até medicina e indústrias.
- Porém, as imagens podem ser afetadas por ruído, distorções e imperfeições que prejudicam sua utilidade.
- A filtragem espacial é uma técnica poderosa usada para melhorar a qualidade e a interpretação de imagens.

Objetivos da Filtragem Espacial

- A filtragem espacial tem como objetivo aprimorar imagens, tornando-as mais adequadas para análise e interpretação.
- Ela pode remover ruídos indesejados, destacar características importantes e realçar informações de interesse.

O que é Filtragem Espacial?

- **Ela envolve a aplicação de operadores matemáticos ou máscaras em uma imagem para processar seus pixels.**
- **O objetivo é alterar a intensidade de cada pixel com base nos valores dos pixels vizinhos, usando uma máscara definida.**
- **Ela é uma parte essencial do processamento de imagens e é usada em diversas áreas, incluindo medicina, visão computacional e fotografia digital.**

Máscaras em Filtragem Espacial

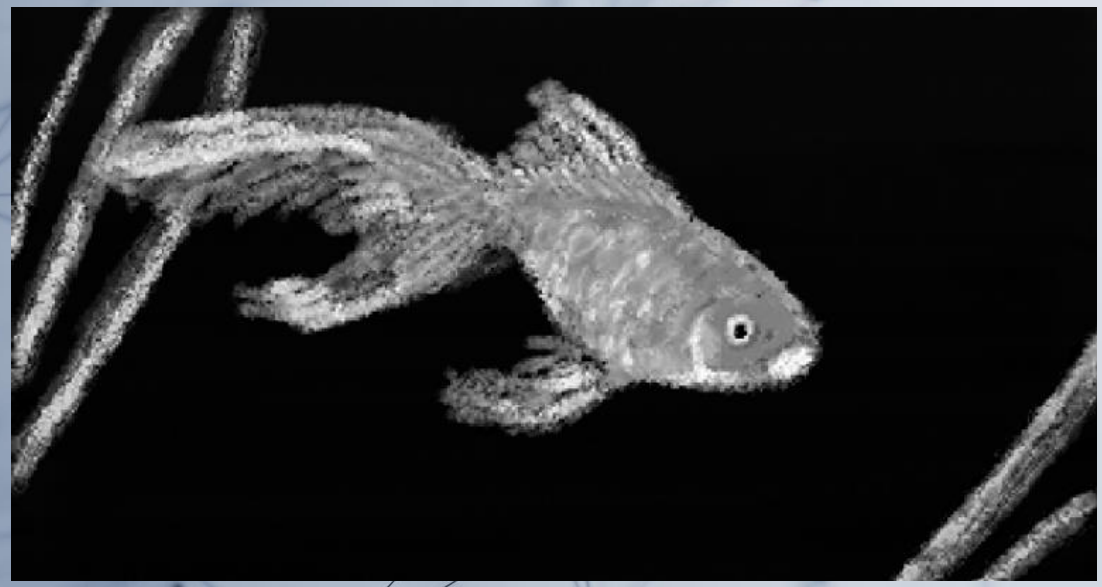
- Máscaras são matrizes de números usadas na filtragem espacial para realizar cálculos em pixels de uma imagem.
- Essas matrizes são colocadas sobre a imagem e deslizam por ela, multiplicando os valores dos pixels pelos valores correspondentes na máscara.

Funcionamento das máscaras

- As máscaras determinam como a filtragem será realizada, influenciando o efeito desejado na imagem.
- Uma máscara é essencialmente uma janela de cálculo que se move pela imagem. Cada elemento na máscara representa um peso atribuído a um pixel específico na vizinhança da janela de cálculo.
- Ao deslizar a máscara pela imagem, calculamos a soma ponderada dos valores dos pixels, resultando em um novo valor para o pixel central na imagem de saída.
- A escolha da máscara e seus valores determina o tipo de filtragem realizado.

Aplicações de Filtragem Espacial

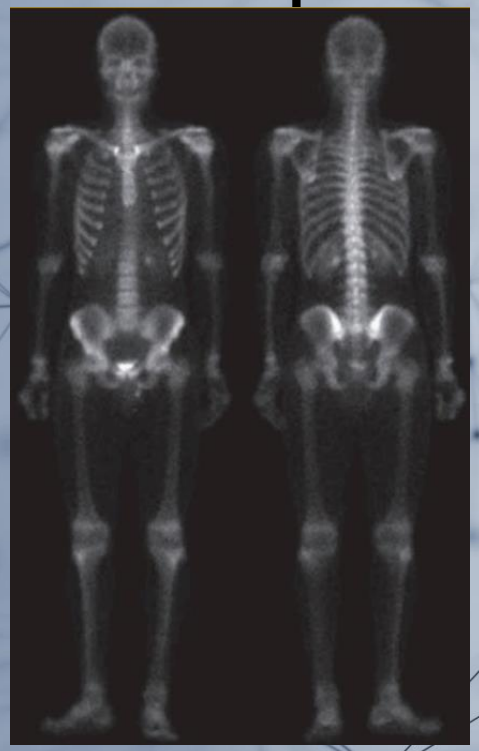
A filtragem espacial é uma técnica versátil com várias aplicações práticas em diversas áreas

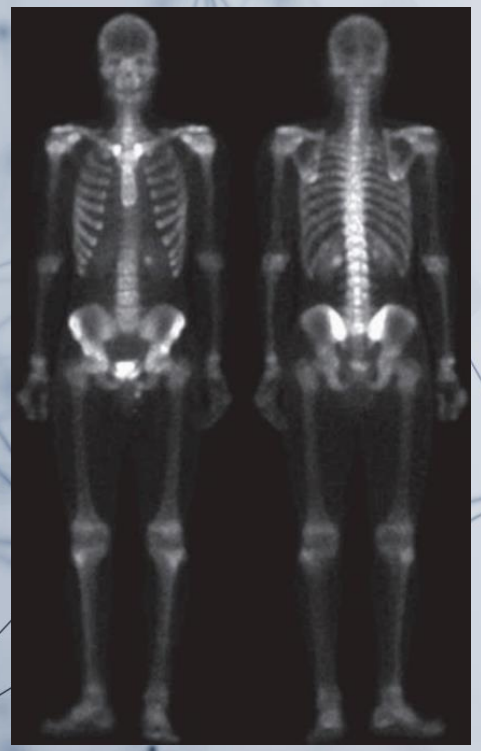


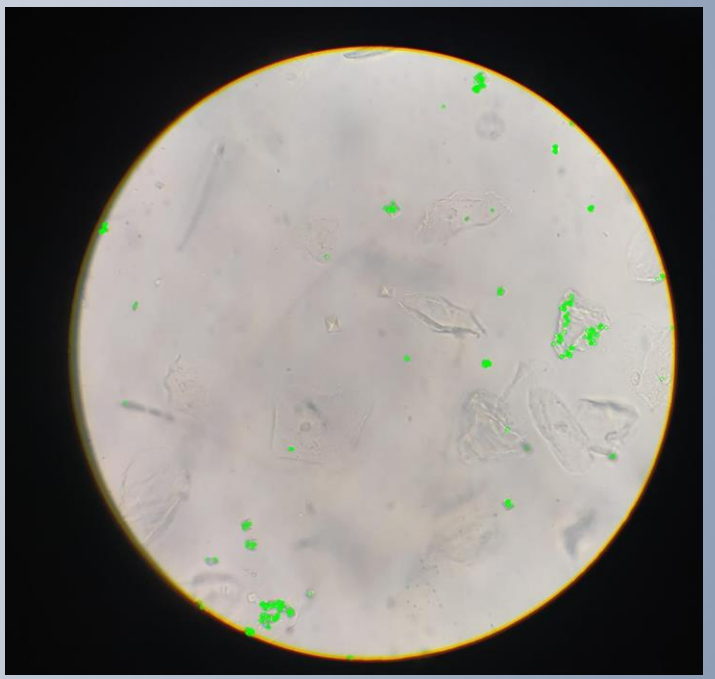
- **Medicina: Melhorias em imagens médicas, como ressonâncias magnéticas e tomografias. A filtragem espacial pode aumentar a clareza e qualidade das imagens para diagnósticos mais precisos.**
- **Fotografia Digital: Redução de ruído em imagens de baixa luminosidade. A filtragem ajuda a eliminar imperfeições e granulações, tornando as fotos mais suaves e nítidas.**

Aplicações de Filtragem Espacial

Exemplos:







Aplicações de Filtragem Espacial

Exemplos:

Matemática da Filtragem Espacial

EXEMPLO: OPERAÇÕES PIXEL A PIXEL

- **Adição/Subtração: Para realizar a adição ou subtração de uma constante a cada pixel da imagem, você simplesmente adiciona ou subtrai essa constante ao valor do pixel.**
- **Multiplicação/Divisão: A multiplicação e a divisão de cada pixel por uma constante podem ser usadas para ajustar o contraste da imagem.**
- **Combinação de Imagens: Você pode realizar operações pixel a pixel que envolvem a combinação de pixels de duas ou mais imagens.**
- **Funções Matemáticas: Você pode aplicar funções matemáticas mais complexas a cada pixel, como a função seno, logaritmo, exponencial, entre outras.**
- **Operações Lógicas: Em imagens binárias (preto e branco), operações pixel a pixel podem envolver operações lógicas, como AND, OR e XOR, para combinar ou modificar áreas de interesse na imagem.**

Matemática da Filtragem Espacial

EXEMPLO: CONVOLUÇÃO

- Definição do Kernel: Começa-se com um pequeno filtro bidimensional chamado "kernel" ou "máscara". O kernel é uma matriz que contém pesos que definem a operação de convolução que será aplicada à imagem. A dimensão do kernel varia de acordo com o tipo de operação que se deseja realizar.
- Posicionamento do Kernel: O kernel é colocado em uma posição inicial na imagem de entrada.
- Multiplicação de Elementos: Para cada posição do kernel, os elementos do kernel são multiplicados pelos valores dos pixels correspondentes na imagem de entrada.
- Soma dos Produtos: Após a multiplicação, os produtos são somados para obter um único valor.
- Armazenamento do Resultado: O resultado da soma é atribuído ao pixel correspondente na imagem de saída.
- Deslocamento do Kernel: O kernel é deslocado para uma nova posição na imagem de entrada, e o processo é repetido até que todas as posições válidas na imagem de entrada tenham sido processadas.
- Imagem Resultante: A imagem resultante contém os valores calculados para cada pixel após a aplicação da convolução. Essa nova imagem é geralmente chamada de "imagem convoluída".

Tipos de Filtros

Filtros de Suavização:

- **Objetivo: Reduzir o ruído e variações bruscas na imagem, criando uma versão mais suave.**
- **Exemplo:**

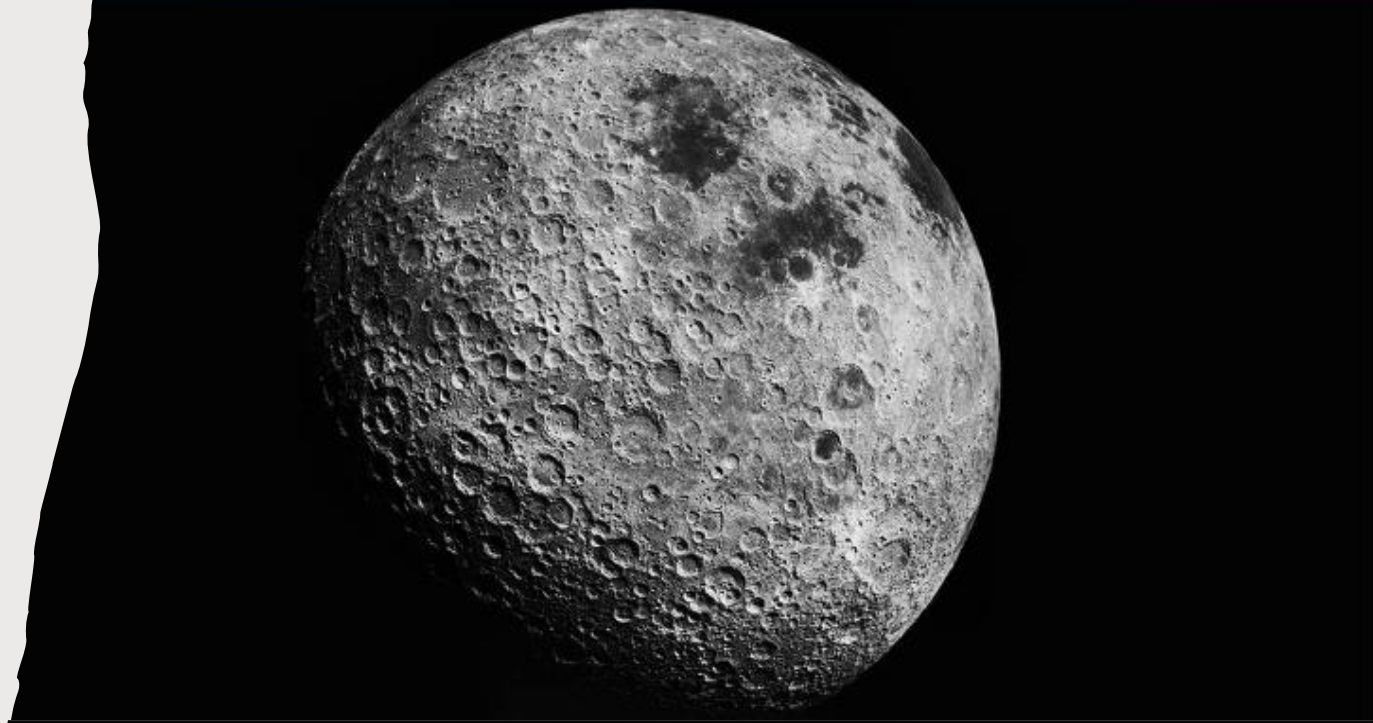
Filtro Gaussiano: Aplica uma função gaussiana para dar peso maior aos pixels centrais.

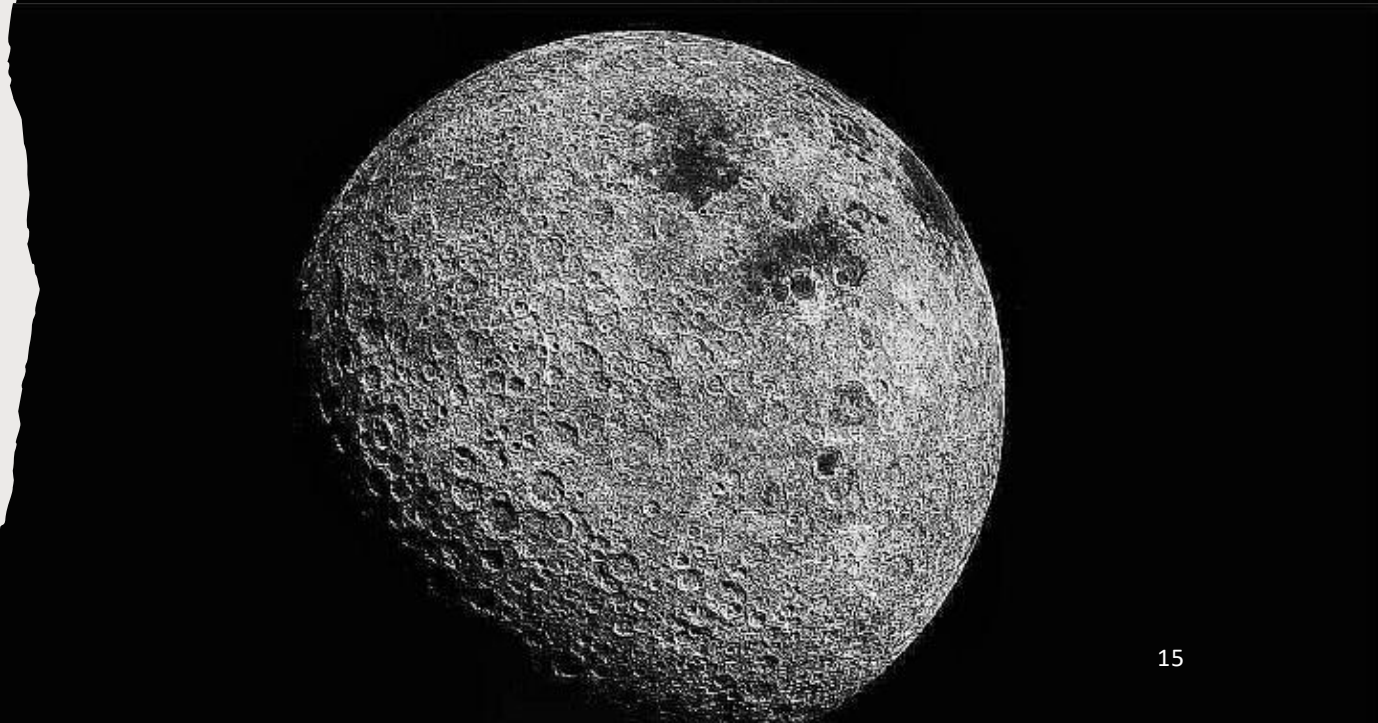
Filtros de Realce:

- **Objetivo: Destacar detalhes e bordas na imagem, tornando-a mais nítida.**

- **Exemplos:**

Filtro Laplaciano: Realça mudanças abruptas na intensidade dos pixels.





Detecção de características:

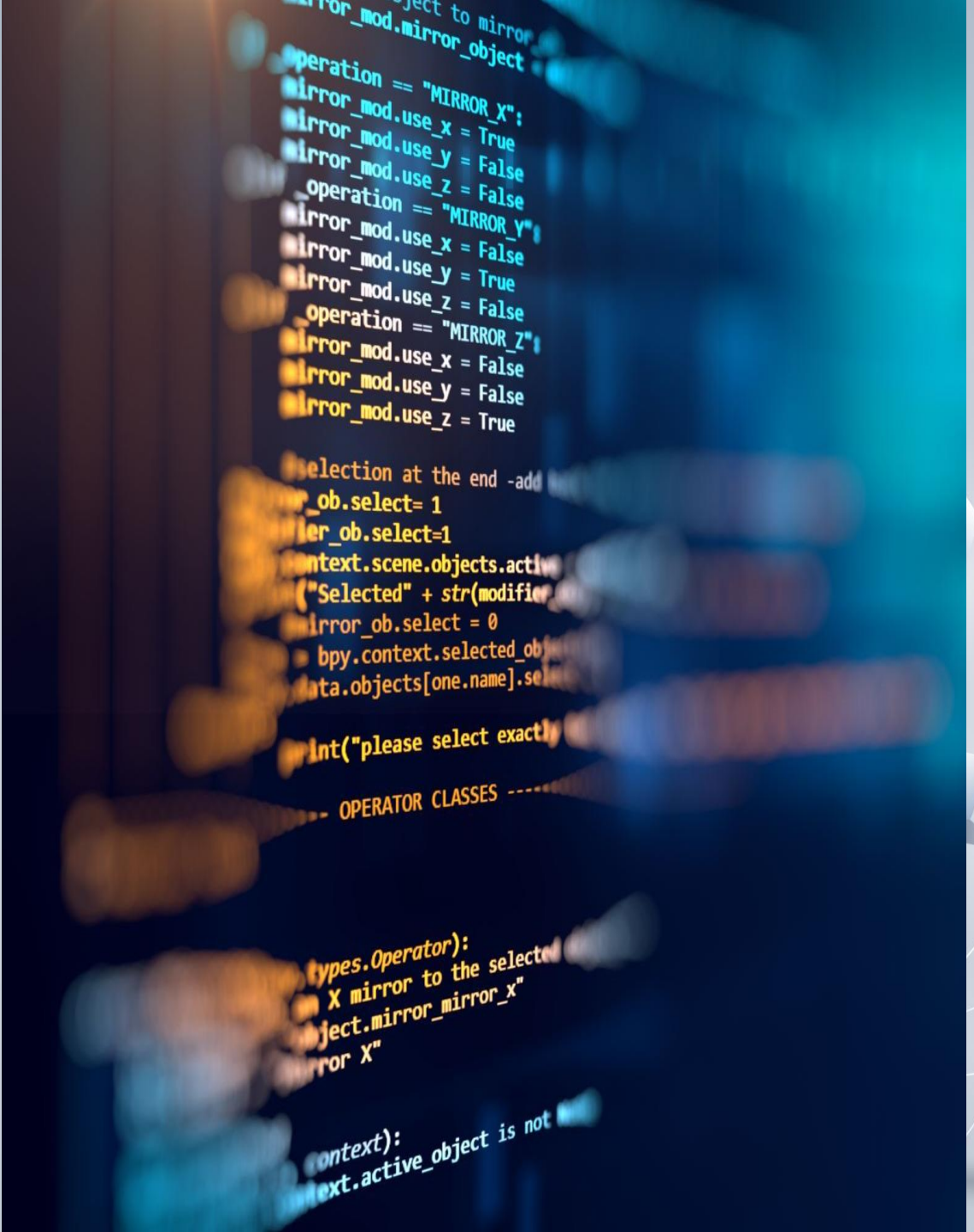
- **Objetivo: Ele é particularmente útil para identificar padrões, texturas ou características específicas em uma imagem.**

- **Exemplo:**

Uma rede neural convolucional (CNN) usará muitos desses filtros diferentes em camadas de convolução para extrair várias características da imagem, permitindo que a rede aprenda a reconhecer objetos, texturas, formas e outros elementos visuais em tarefas de visão computacional.

Exercício!

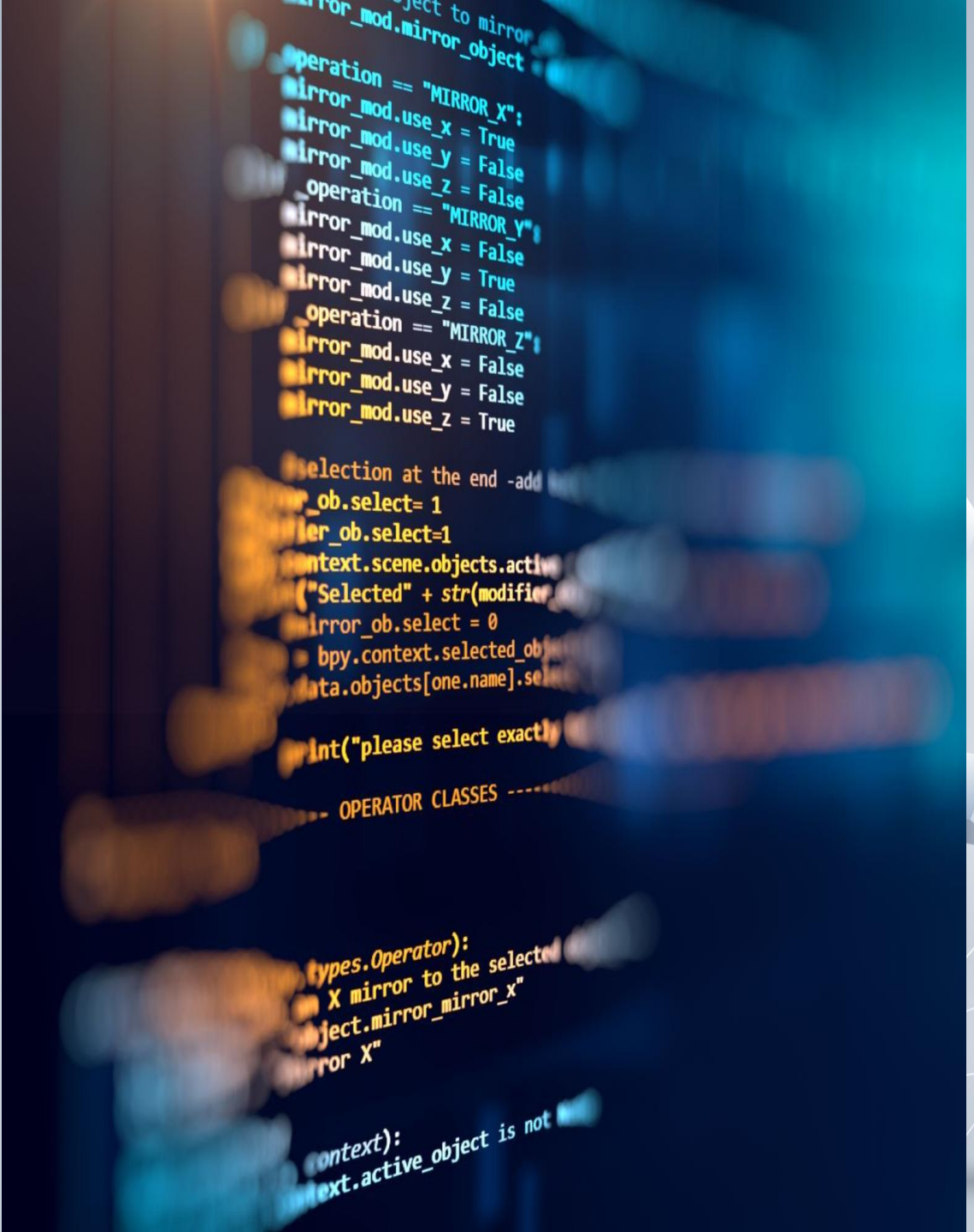
Codificação - Python



Importação de bibliotecas:

```
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
import cv2
import numpy as np
import io
```

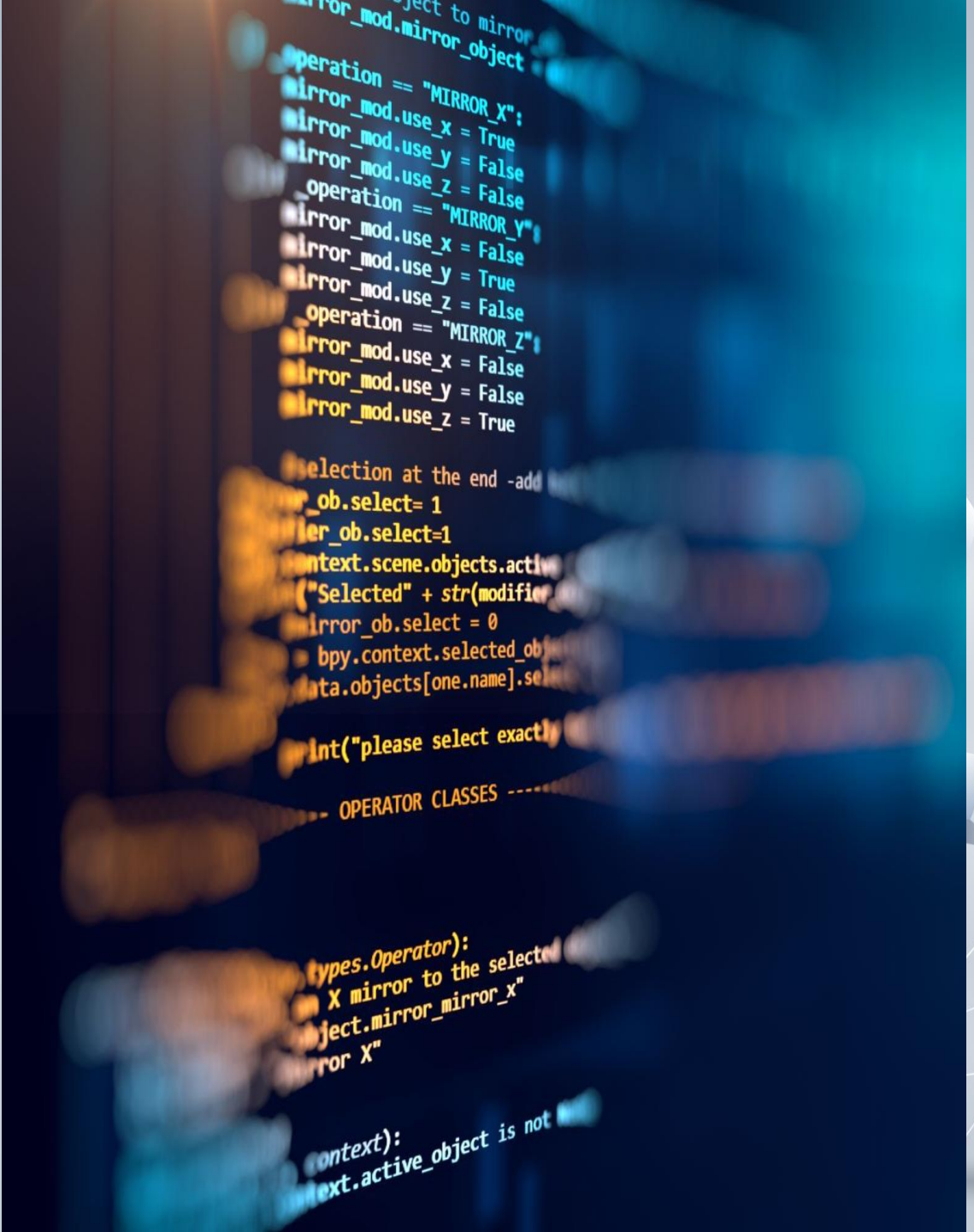
- o Flask para criar o aplicativo da web.
- Bibliotecas para manipular imagens (OpenCV - cv2 e NumPy).
- A biblioteca flask_cors para permitir a comunicação de origem cruzada (CORS) entre domínios diferentes.



Inicialização do servidor

```
app = Flask(__name__)
CORS(app)
```

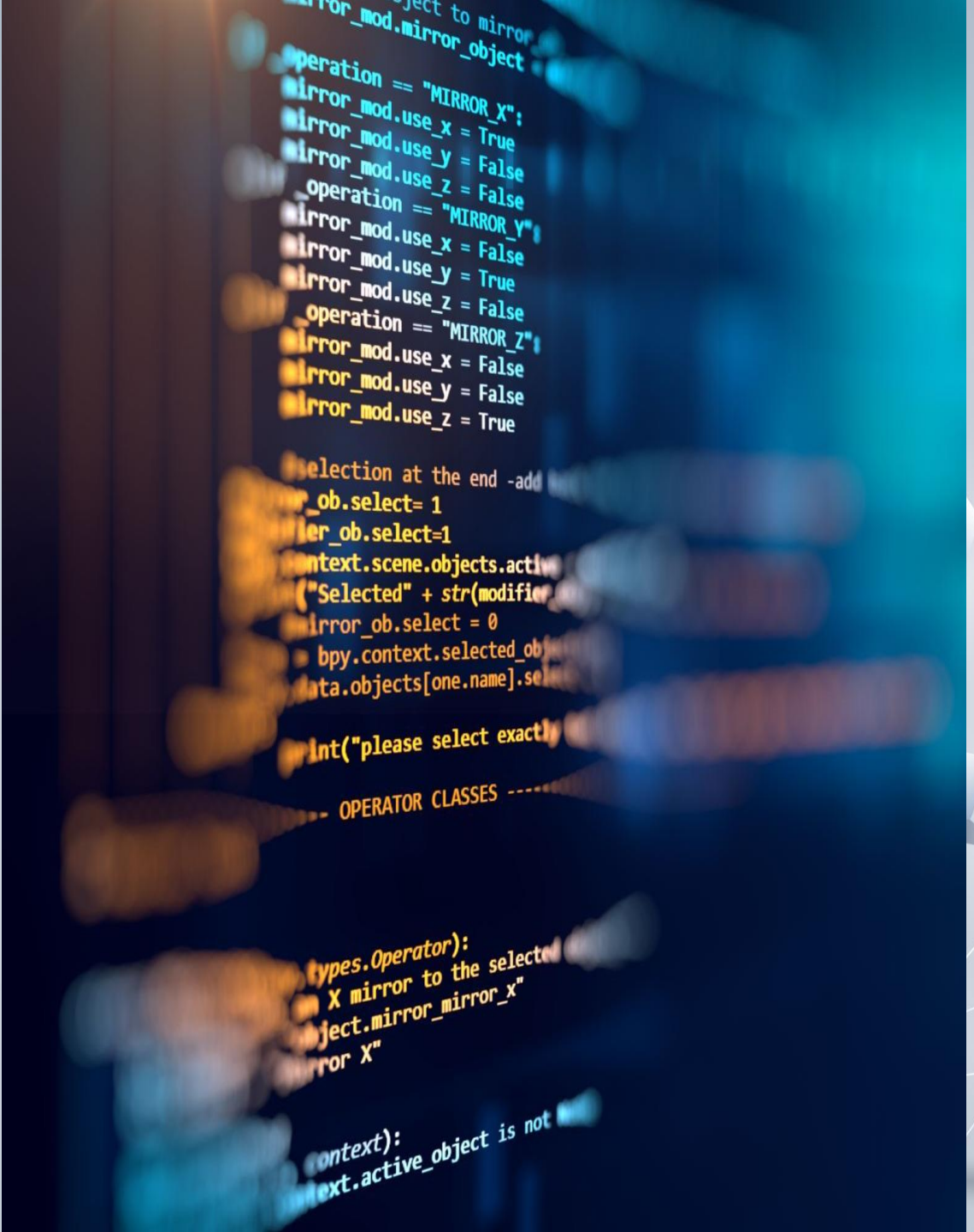
- O aplicativo Flask é inicializado usando Flask(__name__), e o CORS é configurado para permitir comunicação entre origens cruzadas, o que é útil quando o cliente e o servidor estão em domínios diferentes.



Definição de rota POST

```python
@app.route('/process-image', methods=['POST'])
def process_image():
```

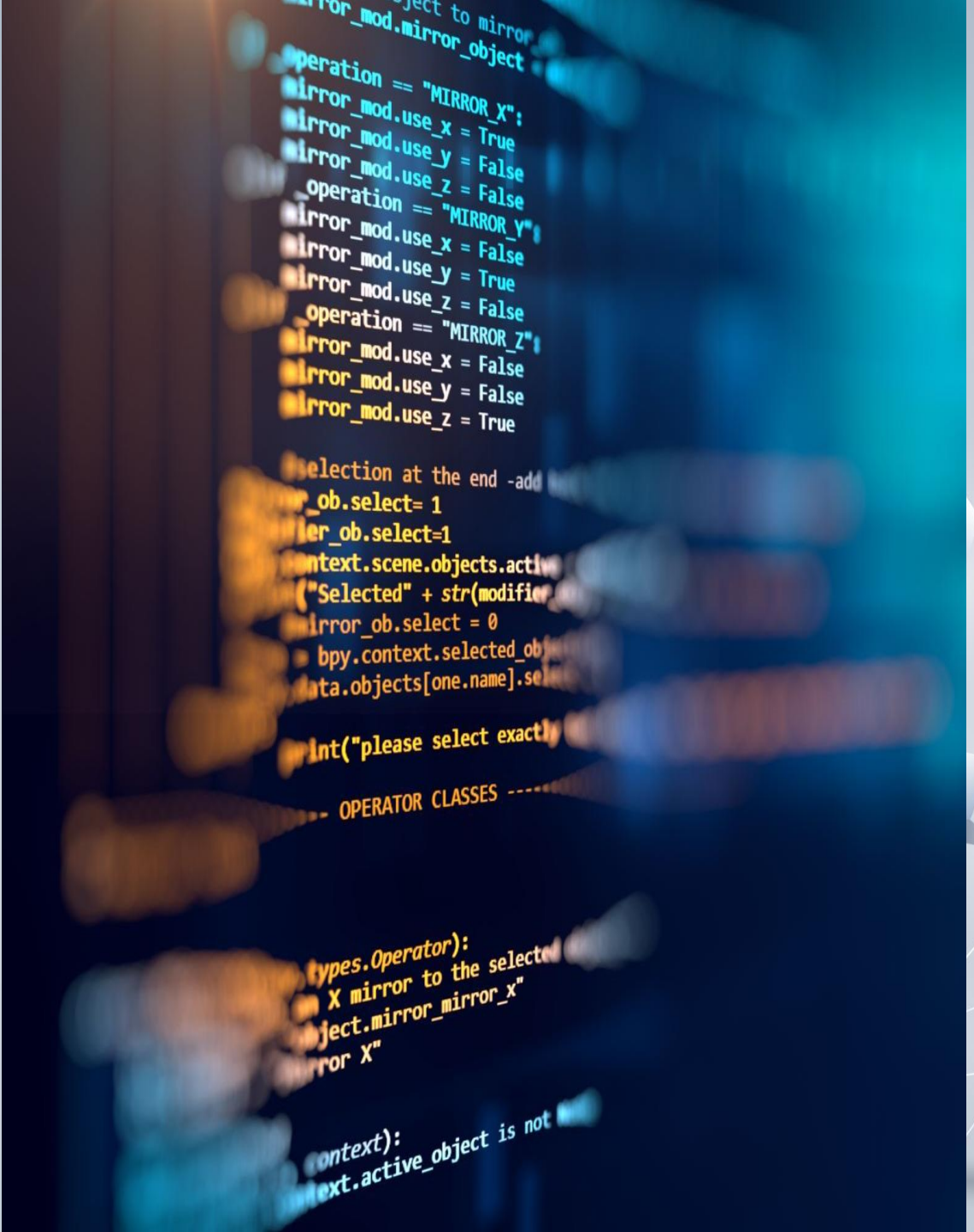
- Esta linha define uma rota chamada /process-image que será acionada quando o servidor receber uma solicitação POST nesta rota. A função process_image() será executada para lidar com a solicitação.



Recebendo a imagem do cliente

```
# Receber a imagem do cliente
image_data = request.files['image']
image = cv2.imdecode(np.frombuffer(image_data.read(), np.uint8), cv2.IMREAD_COLOR)
```

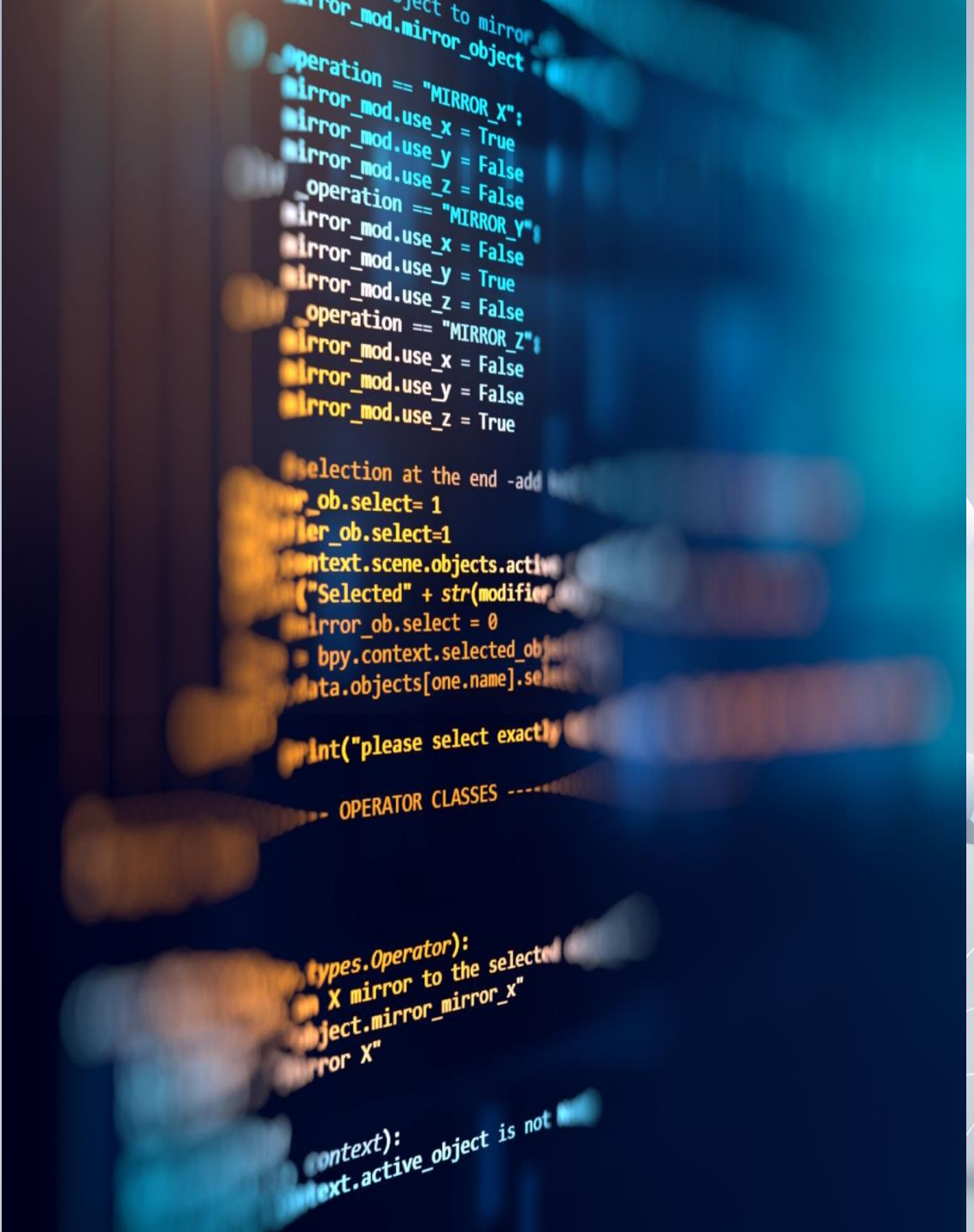
- Neste bloco, o código lê a imagem enviada pelo cliente através da solicitação POST. A imagem é decodificada usando o OpenCV (cv2) e convertida em um formato adequado para processamento.



Recebendo a opção do cliente

```python
# Receber a opção do cliente
selected_option = request.form['option']
```

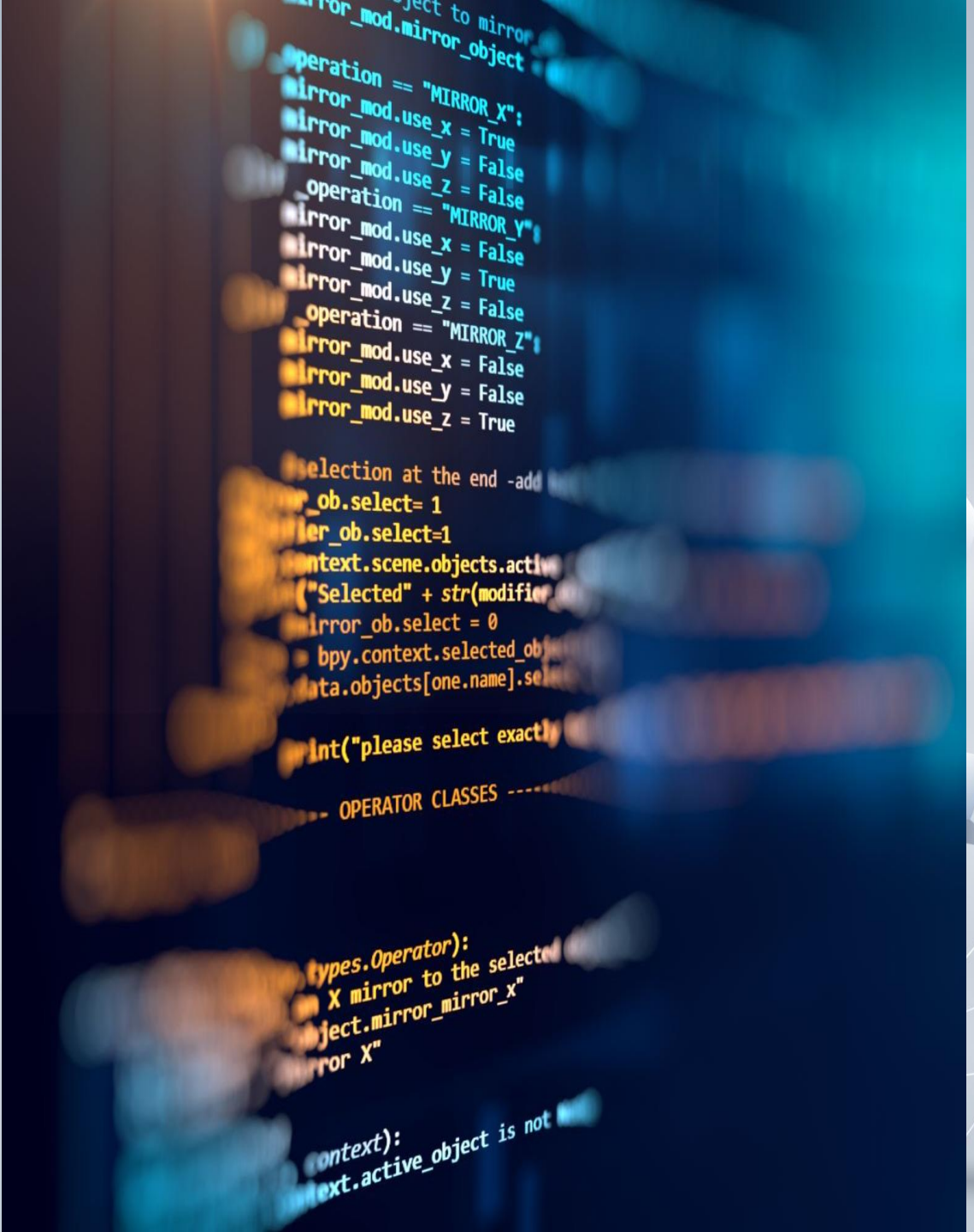
- Aqui, o código obtém a opção selecionada pelo cliente na solicitação POST. O cliente deve fornecer uma opção que será usada para determinar como a imagem será processada.



Realce de imagem

```python
if selected_option == 'enhance':
    # Realçar a imagem
    kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    processed_image = cv2.filter2D(image, -1, kernel)
```

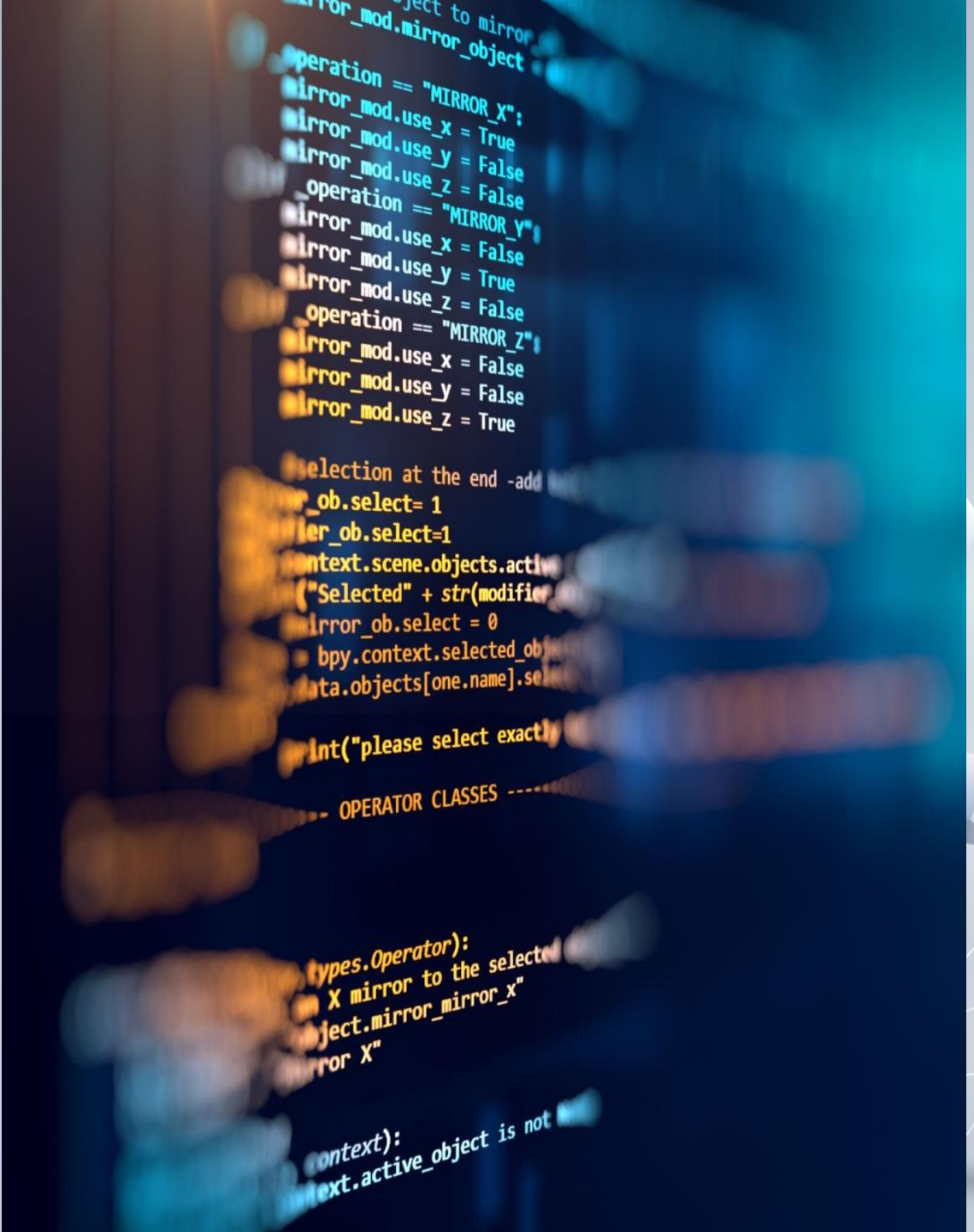
- Esta seção do código verifica a opção selecionada pelo cliente. Se a opção for "enhance", o código aplica um filtro de realce na imagem.



Suavização de imagem

```
elif selected_option == 'smooth':
    # Suavizar a imagem
    processed_image = cv2.GaussianBlur(image, (5, 5), 0)
```

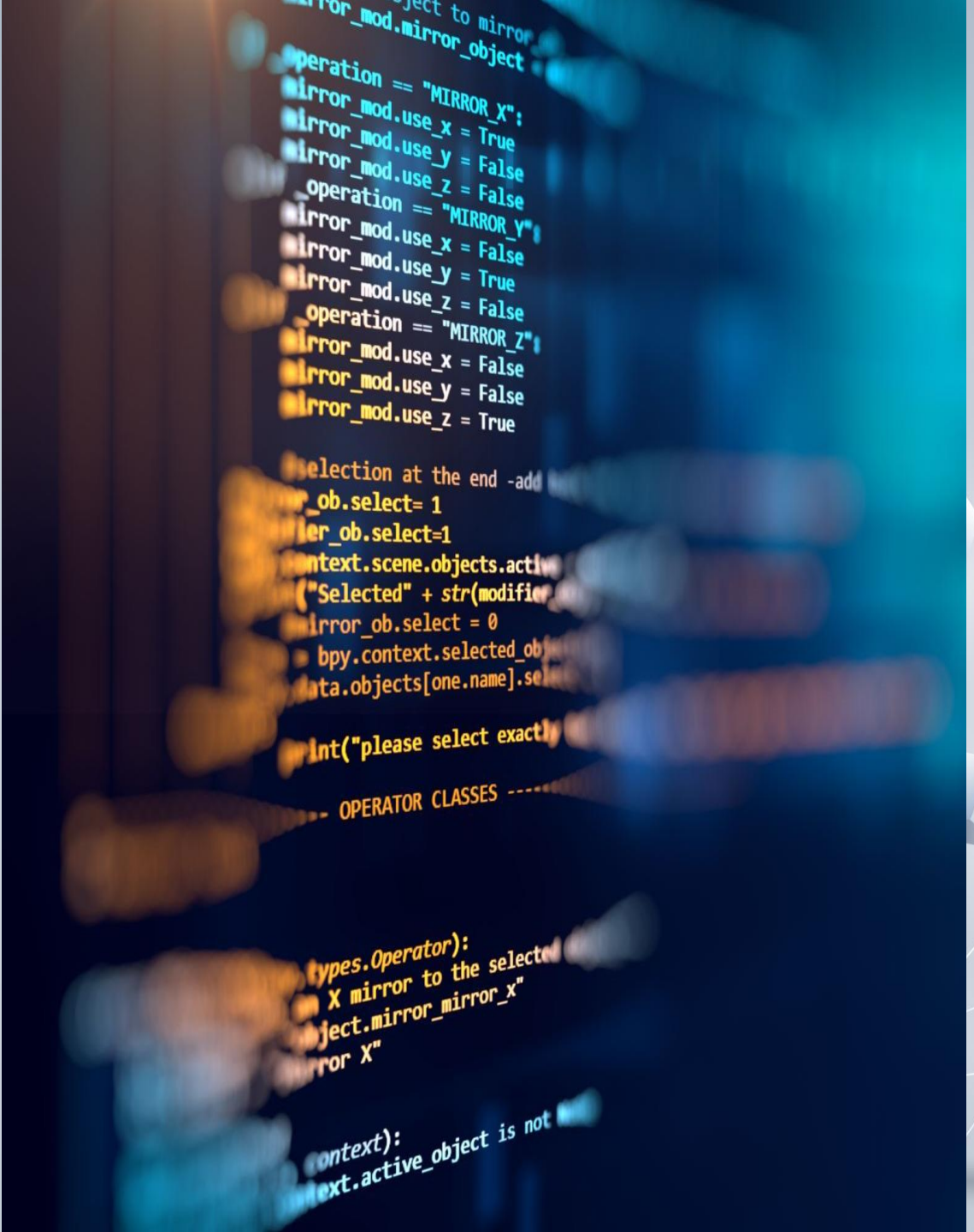
- **Se a opção for "smooth", o código suaviza a imagem usando um filtro Gaussiano.**



Convoluindo a imagem

```python
elif selected_option == 'convolve':
    # Convolução
    kernel = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
    processed_image = cv2.filter2D(image, -1, kernel)
```

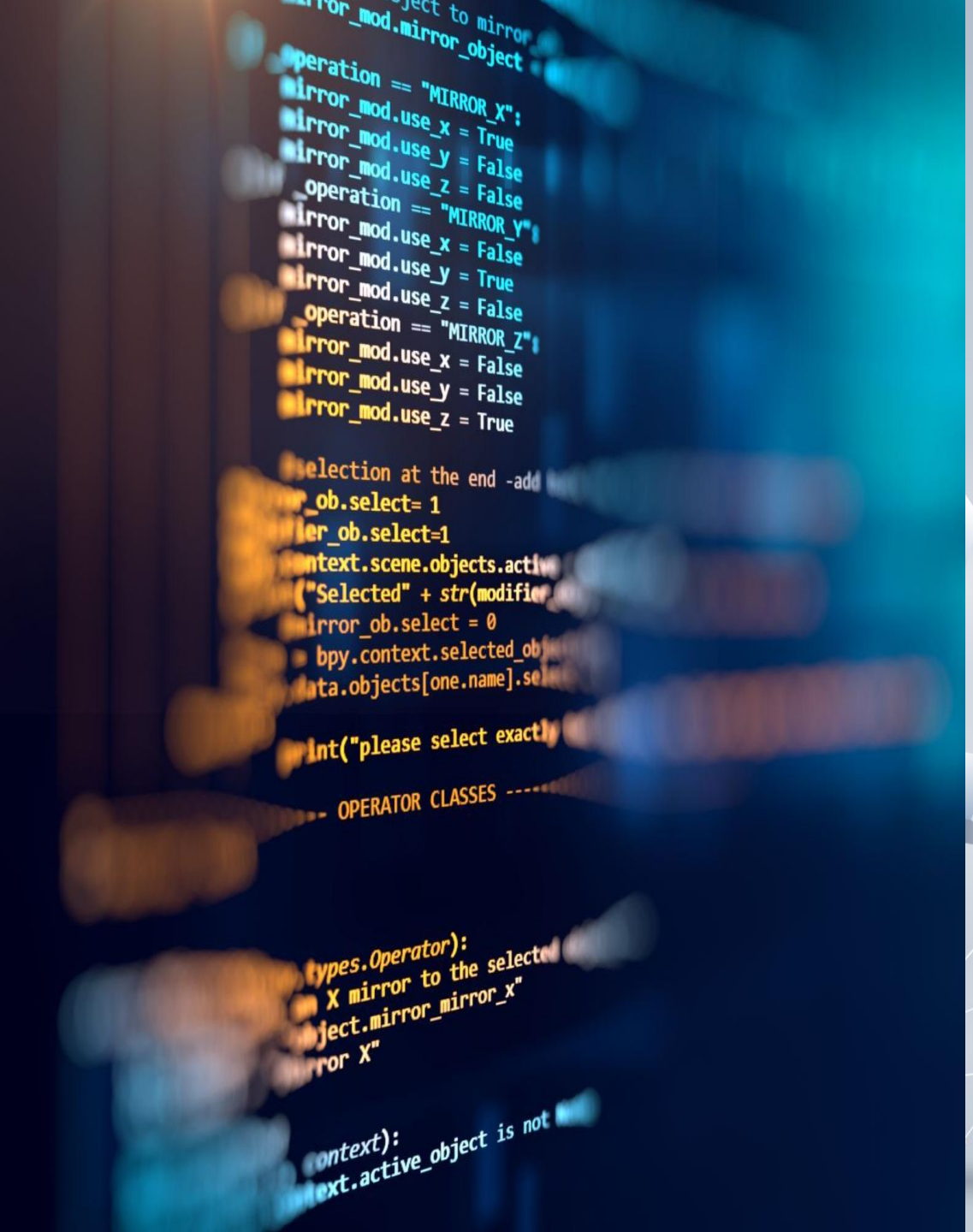
- **Se a opção for "convolve", o código realiza uma operação de convolução na imagem usando um kernel específico.**



Detecção de características

```
elif selected_option == 'detect_features':
    # Detecção de Características (Descritor ORB)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create()
    keypoints, descriptors = orb.detectAndCompute(gray, None)
    processed_image = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0), flags=0)
```

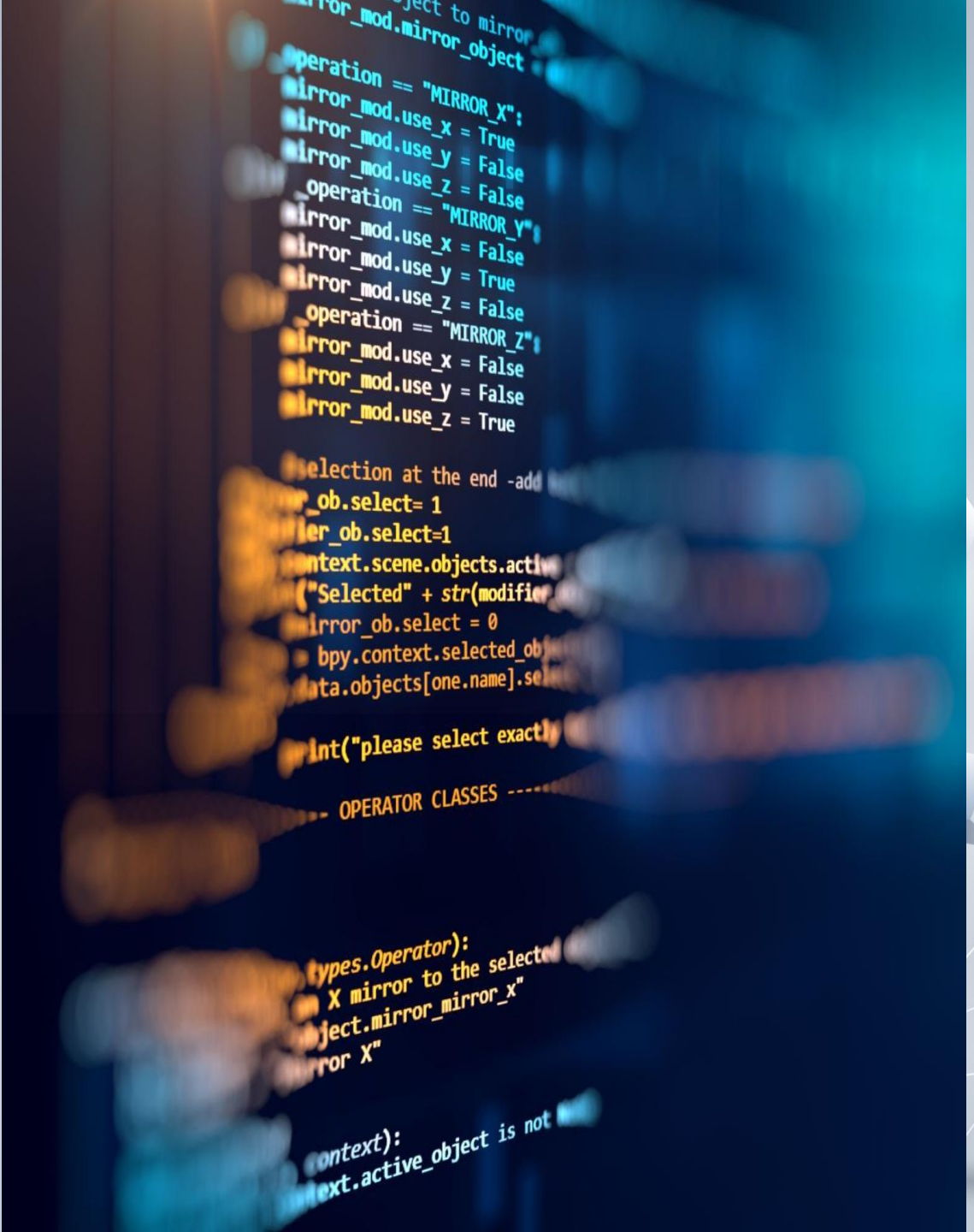
- **Se a opção for "detect_features", o código converte a imagem em escala de cinza, usa o descritor ORB para detectar características e, em seguida, desenha as características na imagem.**



Converter a imagem em bytes

```
# Converter a imagem resultante para um objeto 'bytes'
result_image_bytes = cv2.imencode('.jpg', processed_image)[1].tobytes()
```

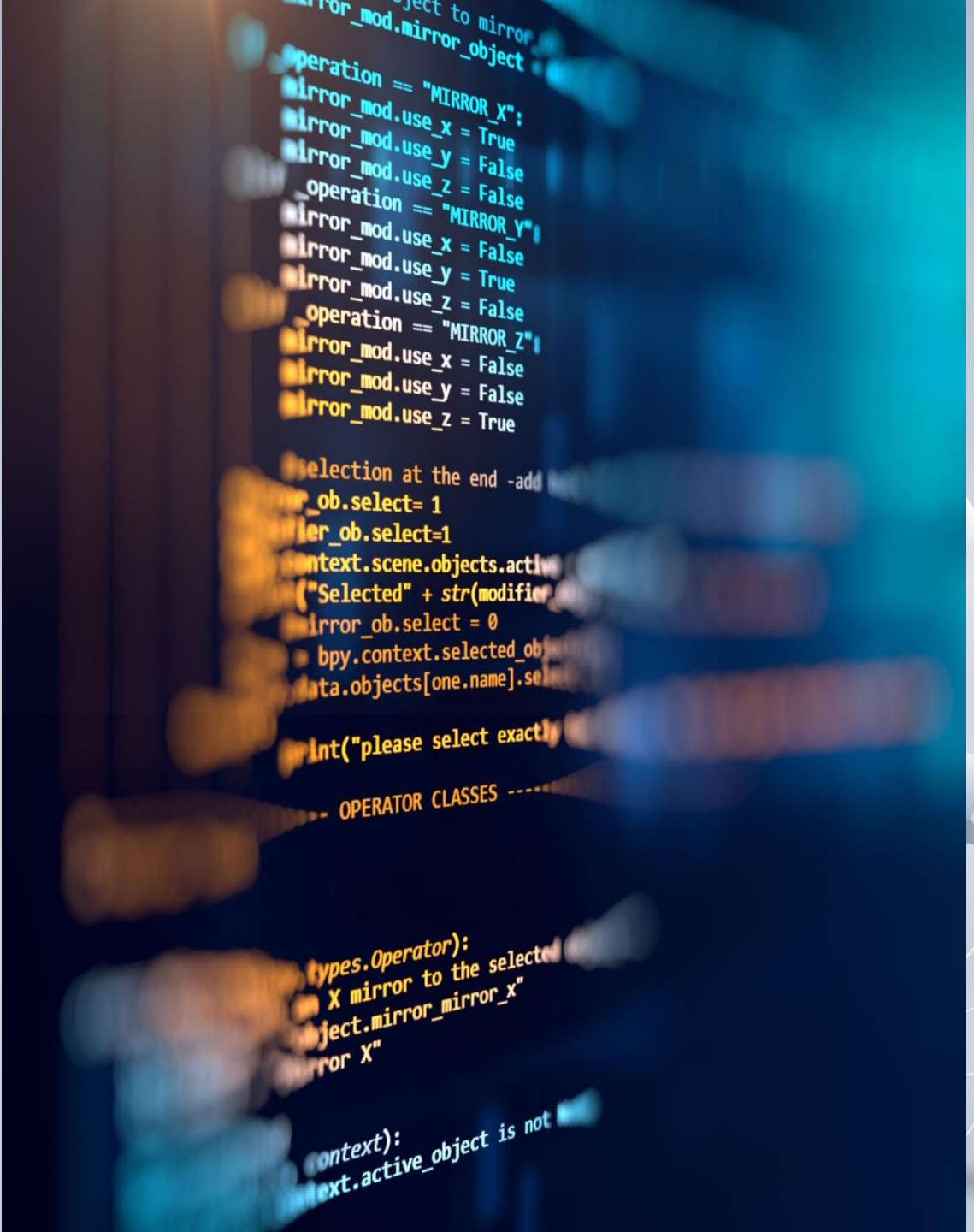
- **Depois de processar a imagem, o código converte a imagem resultante em um objeto de bytes no formato JPEG.**



Retornar o blob

```
# Retornar a imagem como um blob diretamente
return send_file(io.BytesIO(result_image_bytes), mimetype='image/jpeg')
```

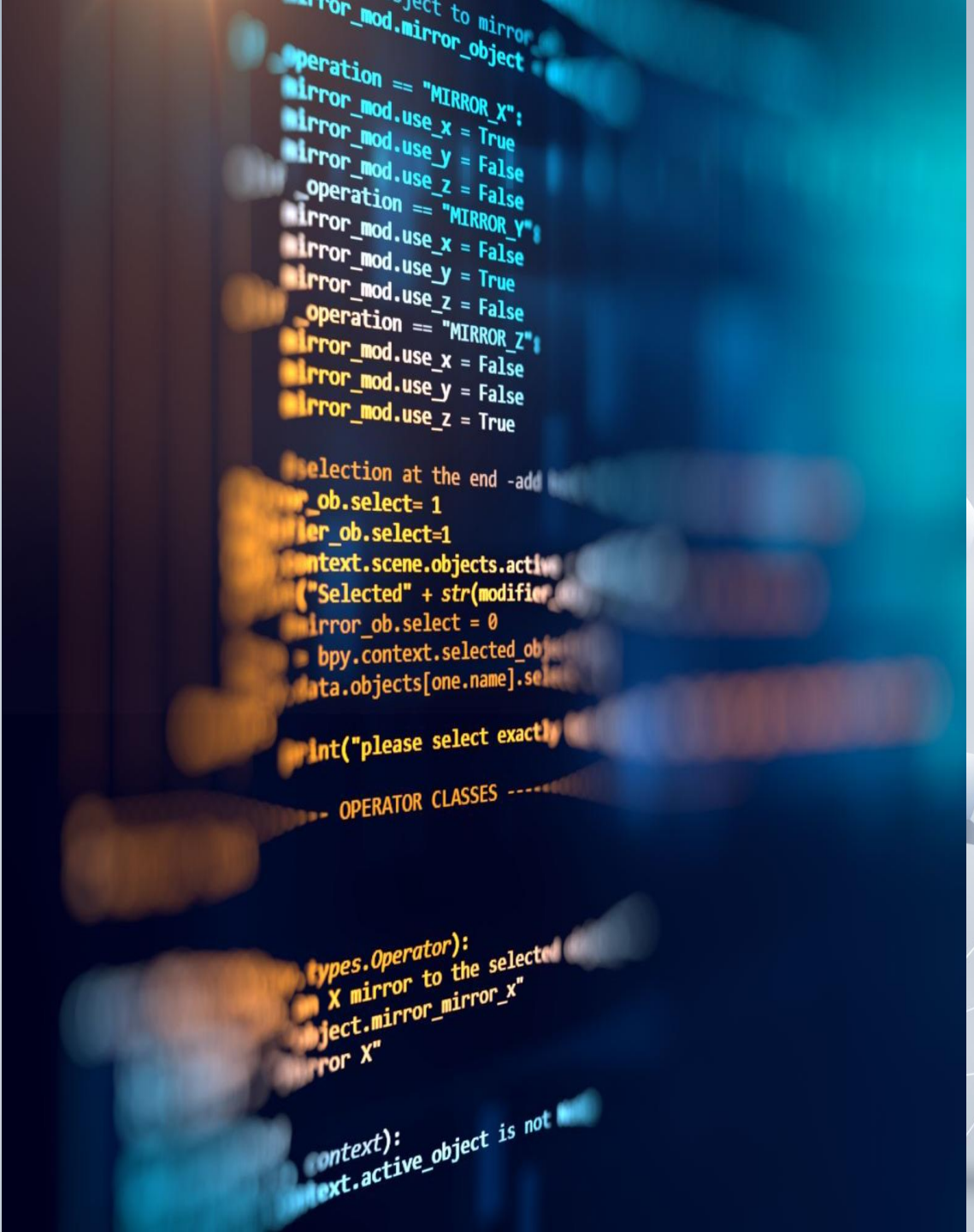
- **Por fim, o código retorna a imagem processada como um arquivo binário diretamente como resposta à solicitação do cliente.**



Tratamento de erro

```
except Exception as e:
    return jsonify({"error": str(e)})
```

- Se ocorrer algum erro durante o processamento da imagem, o código captura a exceção, converte-a em uma mensagem de erro em formato JSON e a retorna como resposta à solicitação do cliente.



Verificação do script

```
if __name__ == '__main__':
    app.run(debug=True)
```

- Por fim, essa parte do código verifica se o script está sendo executado diretamente (não importado como um módulo) e, se for o caso, inicia o aplicativo Flask em modo de depuração. Isso faz com que o servidor Flask seja executado quando o script é executado diretamente.

Desafios e Dicas

- Compreenda o problema e escolha o filtro apropriado.
- Ruído pode distorcer resultados.
- Tamanho do Kernel: Afeta a suavização e realce.
- Teste variações.
- Avalie o Desempenho: Verifique os resultados.
- Aprendizado de Máquina: Em casos complexos.

Dúvidas?

Obrigado!

Yanky Jhonatha Monteiro Fonte Boa

yankyboa@gmail.com

https://github.com/Jhonatha1