Brenda Abreu Almeida, Caio Henrique Braga Teixeira, Guilherme Tadeu Almeida
Sardinha Marques, Héber Stavrakas Gaipo, Pedro Henrique Pereira
Engenharia da Computação 2ºP Noturno

A Aplicação do Cálculo em Visão Computacional: Detecção de Objetos em Imagens Utilizando Python

Divinópolis - MG

27 de janeiro de 2025

Brenda Abreu Almeida, Caio Henrique Braga Teixeira, Guilherme Tadeu Almeida
Sardinha Marques, Héber Stavrakas Gaipo, Pedro Henrique Pereira
Engenharia da Computação 2ºP Noturno

A Aplicação do Cálculo em Visão Computacional: Detecção de Objetos em Imagens Utilizando Python

UNIVERSIDADE ESTADUAL DE MINAS GERAIS

Divinópolis - MG
27 de janeiro de 2025

A Aplicação do Cálculo em Visão Computacional: Detecção de Objetos em Imagens Utilizando Python

Brenda Abreu Almeida Caio Henrique Braga Teixeira
Guilherme Tadeu Almeida Sardinha Marques Héber Stavrakas Gaipo
Pedro Henrique Pereira
Engenharia da Computação 2ºP Noturno

27 de janeiro de 2025

Resumo

Este artigo explora a intersecção entre cálculo e visão computacional, focando na detecção de objetos em imagens. A visão computacional, um campo da inteligência artificial, busca replicar a capacidade humana de interpretar e entender imagens. O processo de detecção de objetos, em particular, se beneficia de conceitos matemáticos, incluindo aqueles estudados em cálculo, como derivadas, integrais e otimização. Este trabalho demonstra como o cálculo é fundamental para o desenvolvimento e aprimoramento de algoritmos de detecção de objetos, utilizando como exemplo prático um código em Python que implementa a detecção de objetos com a arquitetura MobileNet SSD.

Palavras-chave: visão computacional. cálculo. mobilenet. detecção. processamento. python.

1 Introdução

A visão computacional se trata do processo responsável por permitir que sistemas de computadores vejam, identifiquem e entendam o mundo, substituindo a capacidade humana (FENG et al., 2019). Dentro desse campo, a detecção de objetos destaca-se como uma tarefa crucial com ampla aplicação em áreas como veículos autônomos, diagnósticos médicos e robótica. Este artigo explora a intrínseca relação entre a Visão Computacional e o Cálculo, demonstrando como conceitos matemáticos, como derivadas, integrais e otimização, são fundamentais para o desenvolvimento e aprimoramento de algoritmos de detecção de objetos. Para elucidar essa conexão, este trabalho apresenta uma metodologia que combina revisão bibliográfica, seleção de um modelo adequado e implementação prática de um algoritmo de detecção. Os objetivos desta pesquisa são demonstrar a importância do cálculo na Visão Computacional, explicar a aplicação de conceitos específicos do cálculo em redes neurais convolucionais, ilustrar essa aplicação com um exemplo prático utilizando

a arquitetura MobileNet SSD e analisar a relação entre as operações matemáticas e os fundamentos do cálculo. No desenvolvimento, serão abordados os conceitos matemáticos por trás das redes neurais convolucionais, incluindo convoluções, funções de ativação e o processo de backpropagation, além da aplicação de filtros no pré-processamento de imagens. Adicionalmente, o código em Python utilizado para a detecção de objetos será apresentado e explicado, destacando a presença do cálculo em cada etapa. Finalmente, a conclusão sintetizará os principais pontos abordados e reforçará a importância do cálculo para a área da Visão Computacional.

2 Objetivos

2.1 Objetivo Geral

Demonstrar a importância do cálculo no desenvolvimento e aplicação de algoritmos de visão computacional, especificamente na detecção de objetos em imagens.

2.2 Objetivos Específicos

- Explicar como conceitos de cálculo, como derivadas, integrais e otimização, são utilizados no treinamento e na operação de redes neurais convolucionais.
- Ilustrar a aplicação prática desses conceitos por um exemplo de código em Python que implementa a detecção de objetos com a MobileNet SSD.
- Analisar a relação entre as operações matemáticas realizadas pelo código e os conceitos de cálculo subjacentes.

3 Metodologia

Este trabalho utiliza uma abordagem prática para demonstrar a aplicação do cálculo na detecção de objetos em imagens. A metodologia compreende as seguintes etapas:

3.1 Revisão bibliográfica

Inicialmente, foi realizada uma busca de artigos científicos em bases de dados como IEEE Xplore, Mendeley, ScienceDirect, utilizando palavras-chave como “Computer Vision”, “Object Detection”, “Convolutional Neural Network”. Essa revisão serviu para fundamentar a discussão teórica sobre a visão computacional e para selecionar um modelo de detecção de objetos adequado para a demonstração prática.

3.2 Seleção de Modelo e Implementação

Optou-se por utilizar o modelo MobileNet SSD devido à sua eficiência computacional e por ser bem documentado, facilitando a implementação e a compreensão. A implementação foi realizada em Python, utilizando a biblioteca OpenCV para o processamento de imagens e a biblioteca dnn do OpenCV para a interface com o modelo MobileNet SSD.

3.3 Aplicação do Algoritmo

O código implementa o seguinte fluxo de trabalho: carregamento do modelo pré-treinado (arquivos prototxt e caffemodel), pré-processamento da imagem de entrada

(redimensionamento e normalização), execução da detecção de objetos e pós-processamento dos resultados (filtragem por confiança e desenho das bounding boxes na imagem). O código completo é apresentado na seção seguinte.

3.4 Modelo Utilizado

A MobileNet SSD é uma arquitetura de rede neural convolucional projetada para detecção de objetos em tempo real, mesmo em dispositivos com recursos computacionais limitados. Ela combina a arquitetura MobileNet, eficiente em termos de computação, com o algoritmo Single Shot MultiBox Detector (SSD), que permite a detecção de múltiplos objetos em uma única passagem pela rede.

4 Desenvolvimento

A detecção de objetos moderna frequentemente utiliza redes neurais convolucionais (CNNs). Estas redes são compostas por camadas de neurônios interconectados, cujos pesos são ajustados durante o treinamento para aprender padrões nas imagens (FENG et al., 2019). O processo de treinamento envolve a otimização de uma função de custo, que mede a diferença entre as previsões da rede e os valores reais. Aqui, o cálculo desempenha um papel crucial. Algoritmos de otimização, como o gradiente descendente, utilizam derivadas para encontrar o mínimo da função de custo, ajustando iterativamente os pesos da rede.

As CNNs operam aplicando filtros convolucionais à imagem de entrada. A convolução é uma operação matemática que envolve a integral do produto de duas funções, representando o filtro e a imagem (LI et al., 2018). A derivada também é fundamental para o cálculo do gradiente durante a retropropagação, permitindo que a rede aprenda com os erros e ajuste seus pesos eficientemente (LI et al., 2018).

Além disso, o cálculo é utilizado em técnicas de pré-processamento de imagens, como suavização e detecção de bordas. A suavização, por exemplo, pode ser realizada aplicando filtros gaussianos, cuja definição envolve integrais. A detecção de bordas, por sua vez, utiliza derivadas para identificar mudanças abruptas na intensidade da imagem (LI et al., 2018).

O pré-processamento de imagens frequentemente utiliza filtros para modificar ou realçar características específicas. A aplicação de um filtro envolve a operação matemática de convolução, que relaciona diretamente a integral do produto de duas funções. No contexto de processamento de imagens, uma dessas funções representa a imagem em si, enquanto a outra representa o filtro (também chamado de kernel) (LI et al., 2018).

Matematicamente, a convolução discreta 2D entre uma imagem $I(x, y)$ e um filtro $K(i, j)$ de tamanho $(2n + 1) * (2m + 1)$ é definida como:

$$S(x, y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} I(x - i, y - j) \cdot K(i, j)$$

Onde $S(x, y)$ representa o valor do píxel de saída na posição (x, y) após a aplicação do filtro, resultado da convolução naquela coordenada específica da imagem.

$I(x - i, y - j)$ representa o valor do píxel de entrada na posição $(x - i, y - j)$ e I é a função que descreve a imagem de entrada. Ela mapeia a coordenada (x, y) da imagem para um valor de intensidade (por exemplo, brilho em escala de cinza ou valores RGB para imagens coloridas). x e y são as coordenadas do píxel central da região da imagem que está sendo processada pelo filtro e i e j são os deslocamentos em relação ao píxel central.

$K(i, j)$ representa o valor do filtro kernel na posição (i, j), onde K é a função que descreve o filtro. Ela define os pesos que serão aplicados aos pixéus vizinhos durante a convolução e i e j são as coordenadas no filtro.

$\sum$ representa o somatório e indica que os valores nos limites especificados devem ser somados sendo esses valores $i = -n$ até n e $j = -m$ até m. O filtro tem tamanho de $(2n + 1)$ linhas e $(2m + 1)$ colunas.

Diversos filtros podem ser aplicados para diferentes propósitos, cada um projetado para realçar ou suprimir certas características da imagem. Alguns exemplos notáveis, com suas conexões com o cálculo, incluem:

- Filtro Gaussiano (Suavização): Utiliza uma função gaussiana, definida por exponenciais e conceitos de desvio padrão (estatística, intimamente ligada à probabilidade e ao cálculo), para suavizar a imagem, reduzindo ruído e detalhes finos. A função gaussiana em si é definida por uma integral.
- Filtro Laplaciano (Detecção de Bordas): Aproxima a segunda derivada da imagem, realçando regiões com mudanças abruptas de intensidade, que correspondem a bordas. A derivada é um conceito central do cálculo. O filtro Laplaciano é um exemplo de como o cálculo diferencial é aplicado diretamente no processamento de imagens.
- Filtro Sobel (Detecção de Bordas Direcionais): Calcula o gradiente da imagem em direções específicas (horizontal e vertical), permitindo detectar bordas com diferentes orientações. Novamente, o conceito de gradiente vem do cálculo vetorial, parte integrante do cálculo multivariável.

4.1 Exemplo prático

O código a seguir, em Python, demonstra a detecção de objetos utilizando a arquitetura MobileNet SSD e a biblioteca OpenCV:

Primeiro é preciso importar as bibliotecas necessárias (1). **Lembre-se de instalar as bibliotecas necessárias utilizando o comando *pip install*.**

```
# importe os pacotes necessários
import numpy as np  # Para operações numéricas, como manipulação de matrizes e vetores
import argparse     # Para processar argumentos de linha de comando
import cv2          # OpenCV para processamento de imagens
```

Figura 1 – Importando bibliotecas necessárias

Na figura 2 são configurados os argumentos passados para o modelo:

- Imagem a ser analisada.
- Arquivo de texto *prototxt* que define a arquitetura da rede neural e descreve as camadas da rede, suas conexões e parâmetros, sem especificar os pesos aprendidos durante o treinamento.
- O modelo *caffemodel*, que armazena os pesos aprendidos pela rede neural durante o processo de treinamento. São esses pesos que determinam o comportamento da rede e sua capacidade de realizar tarefas como a detecção de objetos.

- Confiança, responsável por estabelecer um limiar usado para filtrar as detecções realizadas pela rede neural. Ele representa a probabilidade mínima para uma detecção ser considerada válida, caso seja estabelecido como 0.2, serão consideradas apenas probabilidades acima de 20%.

Enquanto o arquivo *prototxt* define a estrutura da rede, o *caffemodel* contém a inteligência aprendida pela rede.

```
# constrói o analisador de argumentos e analisa os argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
                help="caminho para a imagem de entrada") # caminho para a imagem de entrada
ap.add_argument("-p", "--prototxt", required=False, default="models/MobileNetSSD_deploy.prototxt.txt",
                help="caminho para o prototxt") # caminho para o prototxt
ap.add_argument("-m", "--model", required=False, default="models/MobileNetSSD_deploy.caffemodel",
                help="caminho para o modelo caffemodel") # caminho para o modelo caffemodel
ap.add_argument("-c", "--confidence", type=float, default=0.2,
                help="probabilidade mínima para filtrar detecções fracas") # probabilidade mínima para filtrar
                 detecções fracas
args = vars(ap.parse_args())
```

Figura 2 – Configurando argumentos de linha de comando

Na figura 3 são declaradas todas as classes com as quais o modelo foi treinado e a gama de cores a ser utilizada na decoração de cada uma das caixas delimitadoras de objetos.

```
# inicializa a lista de rótulos de classe que o MobileNet SSD foi treinado para
# detectar, então gera um conjunto de cores de caixa delimitadora para cada classe
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
           "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
           "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
           "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Figura 3 – Inicializando classes e cores

Na figura 4 o modelo treinado é carregado.

```
# carrega nosso modelo serializado do disco
print("[INFO] carregando modelo...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

Figura 4 – Carregando modelo treinado

Na figura 5 ocorre o pré-processamento da imagem, antes de alimentar a rede neural, aqui ocorrem o redimensionamento e normalização.

1. Redimensionamento e Interpolação

   Redimensionar uma imagem significa alterar suas dimensões (largura e altura), aumentando ou diminuindo o número de píxeis. Quando uma imagem é ampliada, novos píxeis precisam ser criados. Quando é reduzida, alguns píxeis precisam ser descartados. A interpolação é o método usado para determinar o valor desses novos píxeis ou para combinar os valores dos píxeis existentes ao reduzir a imagem.

- Interpolação como Aproximação de Função: Uma imagem pode ser analisada como uma função matemática discreta, onde cada píxel (x, y) possui um valor de intensidade $f(x, y)$. Redimensionar a imagem é como amostrar essa função em novos pontos. A interpolação, então, age como uma forma de aproximar o valor da função nesses novos pontos, com base nos valores conhecidos dos píxeis originais (GONZALEZ; WOODS, 2006, p. 65).
- Métodos de Interpolação e Cálculo: Existem diversos métodos de interpolação, cada um com diferentes graus de complexidade e precisão. Alguns exemplos comuns incluem:
  - Interpolação Vizinho Mais Próximo: Simplesmente copia o valor do píxel original mais próximo. Não envolve cálculos complexos, mas pode resultar em imagens pixelizadas (SZELISKI, 2011, p. 230).
  - Interpolação Bilinear: Calcula a média ponderada dos quatro píxeis vizinhos. Envolve operações aritméticas e ponderação, sendo conceitos básicos do cálculo (SZELISKI, 2011, p. 61).
  - Interpolação Bicúbica: Utiliza um polinômio de terceiro grau para aproximar a função e calcular o valor do novo píxel. Este método, mais sofisticado, produz resultados mais suaves e se baseia diretamente em conceitos de cálculo, como derivadas e polinômios (SZELISKI, 2011, p. 166).

2. Normalização e Operações Aritméticas

   A normalização, neste contexto, refere-se ao processo de ajustar os valores dos píxeis para uma determinada faixa. No código, a linha $blob = cv2.dnn.blobFromImage(...)$ realiza a normalização subtraindo 127,5 de cada valor de píxel e multiplicando o resultado por 0,007843. Isso efetivamente escala os valores dos píxeis para uma faixa aproximadamente entre -1 e 1.

   - Cálculo e Operações Básicas: Embora a normalização neste caso envolva apenas operações aritméticas simples (subtração e multiplicação), ela é fundamental para o bom funcionamento das redes neurais. A normalização ajuda a evitar problemas de instabilidade numérica durante o treinamento e melhora a desempenho da rede. Além disso, o conceito de escala e transformação de dados é relevante em diversos contextos de cálculo.

```
# carrega a imagem de entrada e constrói um blob de entrada para a imagem
# redimensionando para um tamanho fixo de 300x300 pixels e então normalizando-a
# (observação: a normalização é feita pelos autores da implementação do MobileNet SSD)
image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843, (300, 300), 127.5)
```

Figura 5 – Pré-processamento de imagem

Na figura 6 o algoritmo obtém as detecções. Internamente, a rede neural realiza inúmeras operações, incluindo convoluções (integrais), ativações não-lineares (funções deriváveis) e, no treinamento, o processo de backpropagation se baseia fortemente em derivadas parciais para calcular gradientes para otimização da rede.

1. Convoluções (Integrais):

- O que são: as camadas convolucionais são o coração das redes neurais convolucionais (CNNs). Elas aplicam filtros (kernels) à imagem de entrada para extrair características relevantes, como bordas, texturas e padrões (LI et al., 2018, p. 1). A operação de convolução, matematicamente, é uma forma discreta de integral.
- Cálculo na prática: o filtro desliza pela imagem, multiplicando seus valores pelos valores dos píxeis correspondentes e somando os resultados. Essa soma ponderada é equivalente a calcular a integral do produto das duas funções (filtro e imagem) em uma região específica. Embora a implementação computacional utilize somatórios, a base matemática subjacente é a integração.

2. Ativações Não-Lineares (Funções Deriváveis):

   - O que são: após a convolução, uma função de ativação não-linear é aplicada ao resultado. Essa função introduz não-linearidade no modelo, permitindo que a rede aprenda padrões complexos que não podem ser representados por funções lineares.
   - Cálculo na prática: as funções de ativação devem ser deriváveis para que o processo de treinamento, baseado em gradiente, funcione corretamente. Exemplos comuns de funções de ativação incluem a função sigmoide, a função tangente hiperbólica e a ReLU (Rectified Linear Unit). O cálculo diferencial é essencial para calcular as derivadas dessas funções.
   - Por que deriváveis?: A derivada da função de ativação é usada no processo de backpropagation para calcular o gradiente da função de custo em relação aos pesos da rede.

3. Backpropagation e Derivadas Parciais (Gradientes e Otimização):

   - O que é Backpropagation: é o algoritmo central para o treinamento de redes neurais. Ele calcula o gradiente da função de custo (que mede o erro da rede) em relação aos pesos da rede. Esse gradiente indica a direção na qual os pesos devem ser ajustados para minimizar o erro (LILLICRAP et al., 2020).
   - Cálculo na prática: o backpropagation utiliza a regra da cadeia do cálculo diferencial para calcular as derivadas parciais da função de custo em relação a cada peso da rede. Essas derivadas parciais formam o gradiente.
   - Otimização: o gradiente é usado por algoritmos de otimização, como o gradiente descendente, para atualizar os pesos da rede iterativamente. O gradiente descendente "desce" a superfície da função de custo, buscando o ponto de mínimo, que representa o erro mínimo da rede. A otimização é um conceito fundamental do cálculo.

As convoluções (baseadas em integrais), as ativações não-lineares (funções deriváveis) e o backpropagation (que usa derivadas parciais para calcular gradientes) são operações fundamentais nas redes neurais e dependem fortemente de conceitos de cálculo. Embora as bibliotecas de aprendizado de máquina abstraiam a complexidade matemática, entender os fundamentos do cálculo é crucial para compreender o funcionamento e o treinamento das redes neurais.

Na figura 7 a função for gera um loop que vai iterar sobre as detecções e exibi-las. A detecção de objetos produz geralmente as coordenadas da bounding box (caixa

```
# passa o blob pela rede e obtém as detecções e
# predições
print("[INFO] computando detecções de objetos...")
net.setInput(blob)
detections = net.forward()
```

Figura 6 – Obtendo as detecções

delimitadora) em um formato normalizado, relativo ao tamanho da imagem de entrada da rede neural. Para desenhar a bounding box na imagem original, que pode ter dimensões diferentes, é necessário transformar essas coordenadas normalizadas para as coordenadas da imagem original. Esse processo envolve conceitos de geometria analítica e álgebra linear, intimamente relacionados ao cálculo.

1. Coordenadas Normalizadas:

   As coordenadas da bounding box produzidas pela rede neural são geralmente representadas como:

   - $(x\_center\_norm, y\_center\_norm, width\_norm, height\_norm)$

   Onde:

   - x_center_norm e y_center_norm são as coordenadas x e y do centro da bounding box, normalizadas para o intervalo $[0, 1]$. 0 representa a extremidade esquerda/superior da imagem e 1 representa a extremidade direita/inferior.
   - $width\_norm$ e $height\_norm$ são a largura e altura da bounding box, também normalizadas para o intervalo $[0, 1]$.

2. Transformação para Coordenadas da Imagem Original:

   Para desenhar a bounding box na imagem original, com largura w e altura h, as coordenadas normalizadas precisam ser transformadas:

   - $x\_center = x\_center\_norm * w$
   - $y\_center = y\_center\_norm * h$
   - $width = width\_norm * w$
   - $height = height\_norm * h$

   A partir dessas informações, as coordenadas dos cantos superior esquerdo $(startX, startY)$ e inferior direito $(endX, endY)$ da bounding box podem ser calculadas:

   - $startX = int(x\_center - width/2)$
   - $startY = int(y\_center - height/2)$
   - $endX = int(x\_center + width/2)$
   - $endY = int(y\_center + height/2)$

   A conversão para inteiros $(int())$ é necessária porque as coordenadas dos píxeis devem ser números inteiros.

3. Conexão com Geometria Analítica e Álgebra Linear:

    - Escalonamento: A multiplicação das coordenadas normalizadas pela largura e altura da imagem original é uma forma de escalonamento, um conceito fundamental em geometria analítica e álgebra linear. É equivalente a multiplicar um vetor pelas dimensões da imagem, representadas como uma matriz diagonal.
    - Translação: Calcular as coordenadas dos cantos da bounding box a partir do centro e das dimensões envolve translações, outro conceito importante em geometria analítica.
    - Representação Matricial: Todo esse processo de transformação pode ser representado de forma mais compacta e eficiente usando multiplicação de matrizes, um elemento central da álgebra linear. As coordenadas normalizadas e as dimensões da imagem podem ser representadas como vetores e matrizes, e a transformação pode ser realizada com uma única multiplicação de matrizes.

```
# loop sobre as detecções
for i in np.arange(0, detections.shape[2]):
    # extrai a confiança (ou seja, probabilidade) associada à
    # predição
    confidence = detections[0, 0, i, 2]
    # filtra detecções fracas, garantindo que a `confiança` seja
    # maior que a confiança mínima
    if confidence > args["confidence"]:
        # extrai o índice do rótulo de classe das `detecções`,
        # então calcula as coordenadas (x, y) da caixa delimitadora para
        # o objeto
        idx = int(detections[0, 0, i, 1])
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
```

Figura 7 – Loop para exibir as detecções

Na figura 8 o modelo é configurado para exibir na tela as informações obtidas.

```
        # exibe a predição
        label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
        print("[INFO] {}".format(label))
        cv2.rectangle(image, (startX, startY), (endX, endY), COLORS[idx], 2)
        y = startY - 15 if startY - 15 > 15 else startY + 15
        cv2.putText(image, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
        cv2.imwrite("images/output/image.jpg", image)
```

Figura 8 – Exibindo as informações na tela

- *label*: estabelece o formato de texto que vai imprimir a informação, sendo "*informação: probabilidade com duas casas decimais*", sendo a informação capturada no escopo de classes através do índice passado pelo algoritmo e a probabilidade sendo o valor multiplicado por cem, para aparecer no formato 100,00%.

- Essa variável *label* é impressa no prompt de comando.
- Gera-se na imagem o retângulo que vai delimitar o objeto, coordenadas de início e fim do retângulo passadas por $(startX, startY)$ e $(endX, endY)$, a cor do retângulo definida por $COLORS[idx]$.
- A variável y armazena a coordenada final onde o texto de informação vai ser inserido com o retângulo que delimita o objeto.
- Então a biblioteca $cv2$ insere o texto *label* na imagem *image* nas coordenadas $(startX, y)$ com a fonte $cv2.FONT\_HERSHEY\_SIMPLEX$ no tamanho 0.5 e na mesma cor do retângulo.
- A biblioteca $cv2$ gera um arquivo com essa nova imagem que contém essas informações.

O programa é encerrado através das linhas de código presentes na figura 9. É exibida na tela a imagem gerada pelo programa e o programa aguarda que qualquer tecla seja pressionada, concluindo de vez a execução do programa.

```
# mostra a imagem de saída
cv2.imshow("Output", image)
cv2.waitKey(0)
```

Figura 9 – Encerrando o programa

Embora o código em si não execute cálculos complexos diretamente, ele se baseia em bibliotecas e algoritmos fundamentados em conceitos de cálculo. A própria arquitetura da rede neural convolucional, MobileNet SSD, é construída com base em operações como convolução (sendo uma integral) e utiliza cálculo diferencial para o processo de treinamento (backpropagation com gradiente descendente).

A manipulação de imagens, mesmo em etapas que parecem simples como redimensionamento, pode envolver interpolações que, em sua essência, são aproximações de funções — um conceito central no cálculo. A detecção de bordas, muitas vezes um pré-processamento para detecção de objetos, usa derivadas para encontrar mudanças abruptas nos valores dos píxeis.

Portanto, mesmo que o código não realize integrais ou derivadas explicitamente, ele se beneficia da teoria e das ferramentas fornecidas pelo cálculo para realizar a tarefa de detecção de objetos.

O programa é executado através do prompt *python main.py -i images/image.jpg* que passa a imagem *image.jpg* no diretório local *images* como argumento $-i$ na execução *python* do arquivo *main.py*.

5 Conclusão

Este artigo demonstrou a intrínseca relação entre o cálculo e a visão computacional, especificamente no contexto da detecção de objetos. Através da análise do código Python que implementa a MobileNet SSD, evidenciou-se como conceitos fundamentais do cálculo, como derivadas, integrais e otimização, permeiam as operações realizadas por redes neurais

convolucionais. A discussão sobre os métodos de interpolação utilizados no redimensionamento de imagens, a normalização dos dados e as operações matemáticas intrínsecas às convoluções, funções de ativação e backpropagation reforçam a importância do cálculo para o processamento e interpretação de imagens. A implementação prática, exemplificada com a detecção de objetos em imagens utilizando a MobileNet SSD, consolidou a conexão entre a teoria matemática e sua aplicação na visão computacional. Portanto, este trabalho não apenas cumpriu seus objetivos de demonstrar a importância do cálculo nessa área, mas também forneceu um guia prático para a compreensão das bases matemáticas subjacentes aos algoritmos de visão computacional. A capacidade de conectar esses conceitos teóricos com a prática da programação é essencial para o desenvolvimento de soluções inovadoras e eficientes em visão computacional, impulsionando avanços em áreas como robótica, veículos autônomos e muitas outras.

Exemplo:

Figura 10 – Imagem de Entrada (FERREIRA, 2024)

Figura 11 – Imagem de Saída

Referências

FENG, X. et al. Computer vision algorithms and hardware implementations: A survey. *Integration*, v. 69, p. 309–320, 2019. ISSN 0167-9260. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167926019301762>. Mencionado 2 vezes nas páginas 2 e 4.

FERREIRA, S. *A paz no trânsito começa por você*. 2024. Acesso em: 22/01/2025. Disponível em: <https://mobilidade.estadao.com.br/wp-content/uploads/2023/04/Sabrina-Ferreira_Transito-39.jpg>. Mencionado na página 13.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006. ISBN 013168728X. Mencionado na página 7.

LI, Y. et al. Research on a surface defect detection algorithm based on mobilenet-ssd. *Applied Sciences*, v. 8, n. 9, 2018. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/8/9/1678>. Mencionado 2 vezes nas páginas 4 e 8.

LILLICRAP, T. P. et al. Backpropagation and the brain. *Nature Reviews Neuroscience*, v. 21, 2020. ISSN 1471-0048. Disponível em: <https://doi.org/10.1038/s41583-020-0277-3>. Mencionado na página 8.

SZELISKI, R. *Computer vision algorithms and applications*. London; New York: Springer, 2011. ISBN 9781848829343 1848829345 9781848829350 1848829353. Disponível em: <http://link.springer.com/book/10.1007%2F978-1-84882-935-0>. Mencionado na página 7.