# Visão computacional

**Juliana Patrícia Detroz**
**Orientador: André Tavares Silva**

# Visão computacional

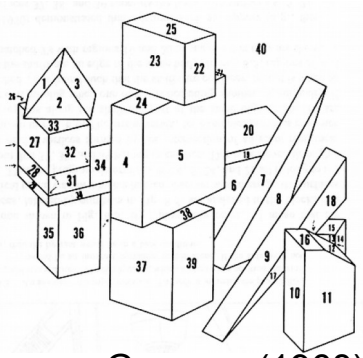
- Tentativa de **replicar** a habilidade humana da visão através da **percepção e entendimento** de uma **imagem**;
- Fazer o computador **entender** imagens e vídeos;
- Obter **informações** a partir de imagens;

- Reconhecer
- Identificar
- Detectar

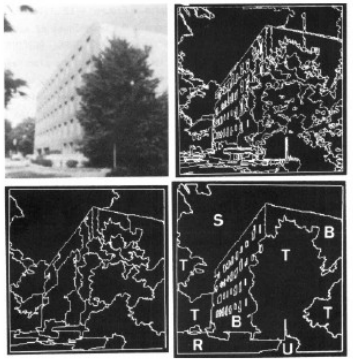
} Objetos na cena

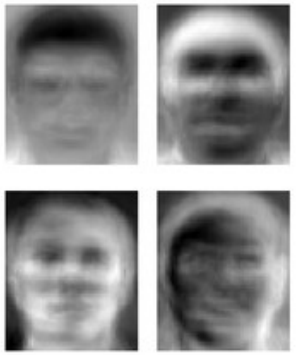
- Alta complexidade
- **Desafio**: como o cérebro processa imagens?

# Visão computacional


Guzman (1968)

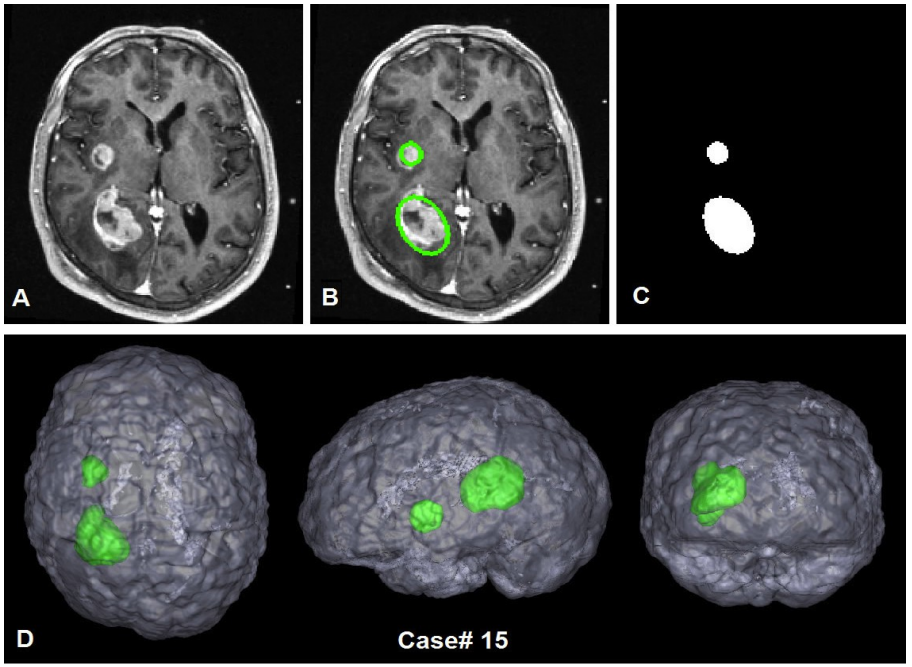
- **1960**: subproblema de **IA**
  - 1966 - Marvin Minsky – Máquina descrever o que vê
  - Interpretação de **mundos sintéticos**
- **1970**: progresso na interpretação de **imagens específicas**
- **1980**: tentativas com redes neurais;
  - Foco em **geometria** e rigor matemático
- **1990**: detecção de **faces**
  - Análise estatística
- **2000**: popularização
  - Métodos de reconhecimento mais **abrangentes**
  - Disponibilização de **bases de dados** anotadas maiores
  - Processamento em **vídeo**


Ohta Kanade (1978)


Turk and Pentland (1991)

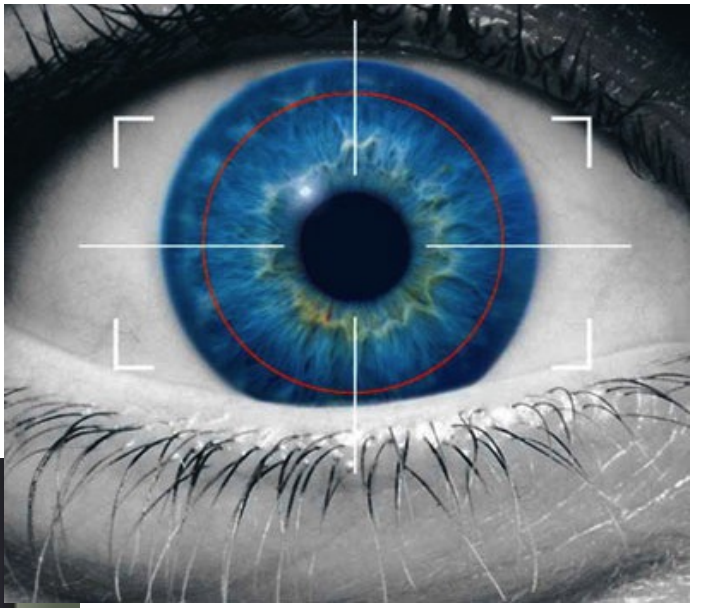
# Visão computacional

- Aplicações – Saúde:



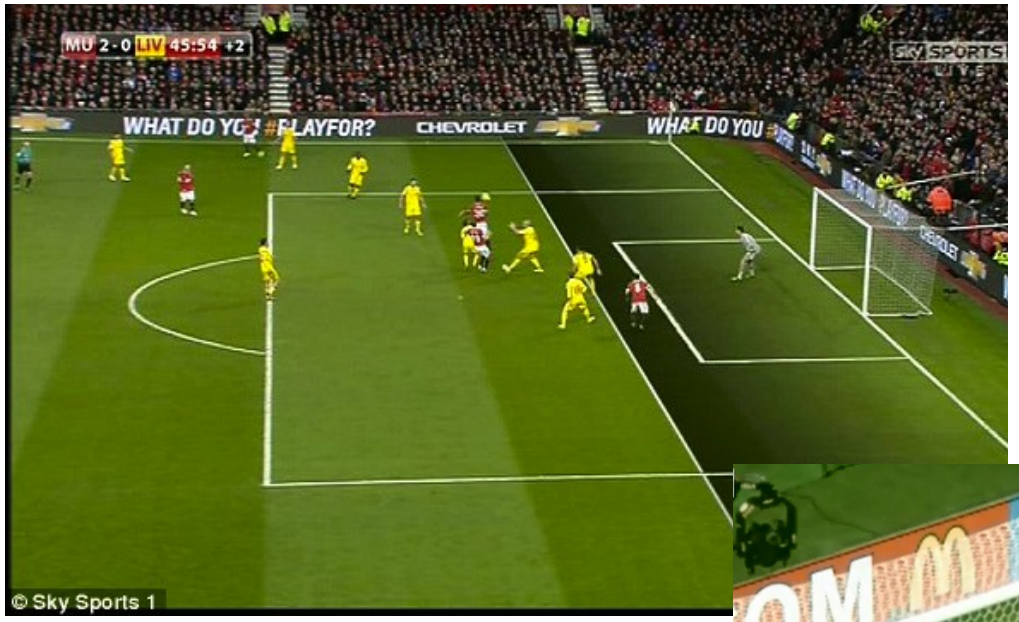

# Visão computacional

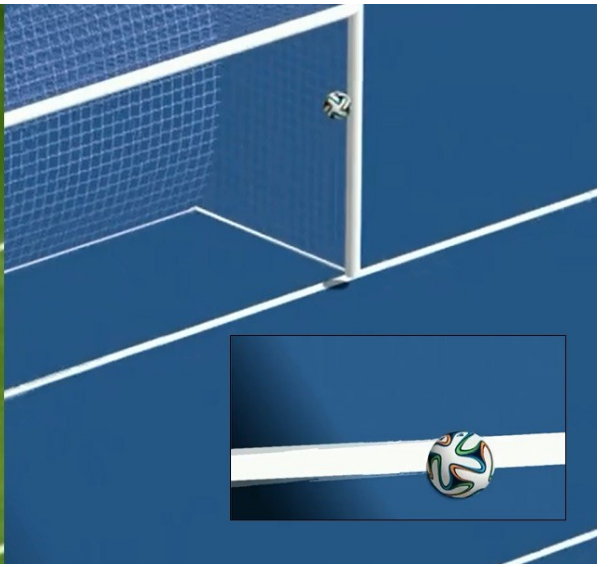
- Aplicações – Controle de acesso:

# Visão computacional

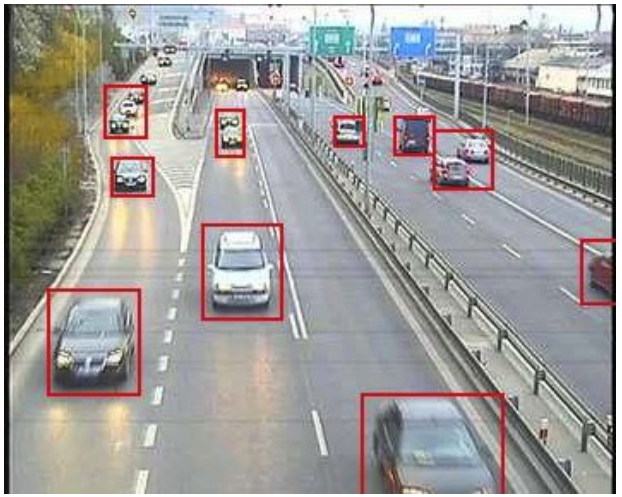
- Aplicações – Esportes:

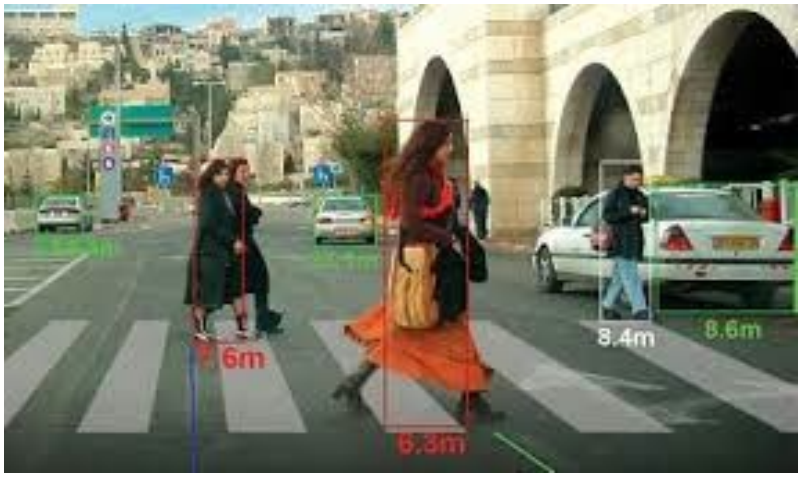


# Visão computacional

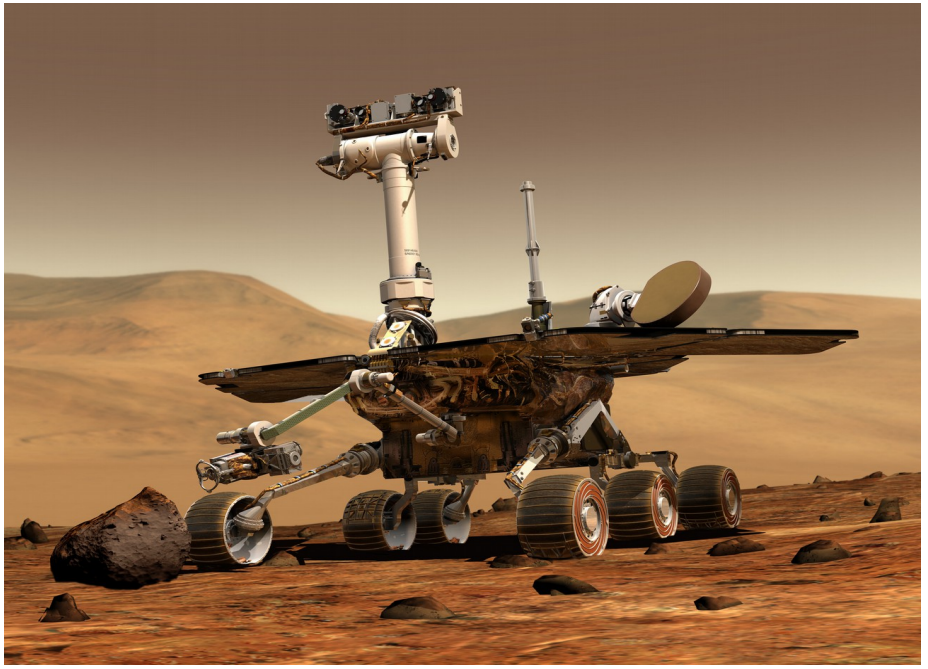
- Aplicações – controle de tráfego:





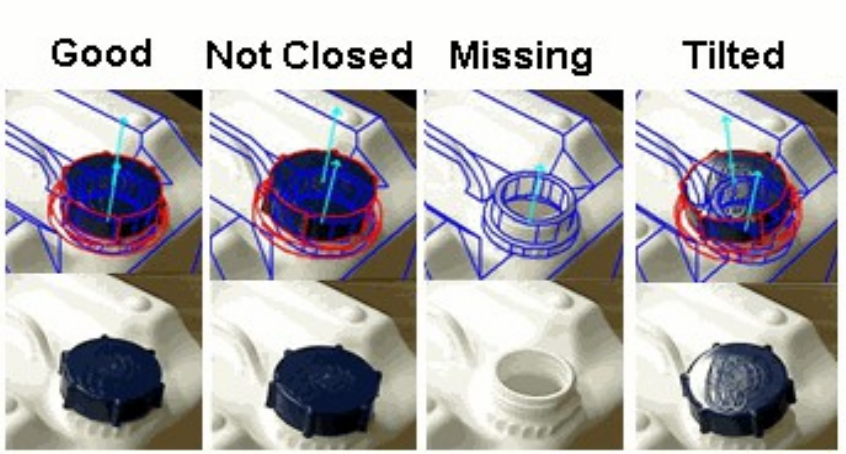
# Visão computacional

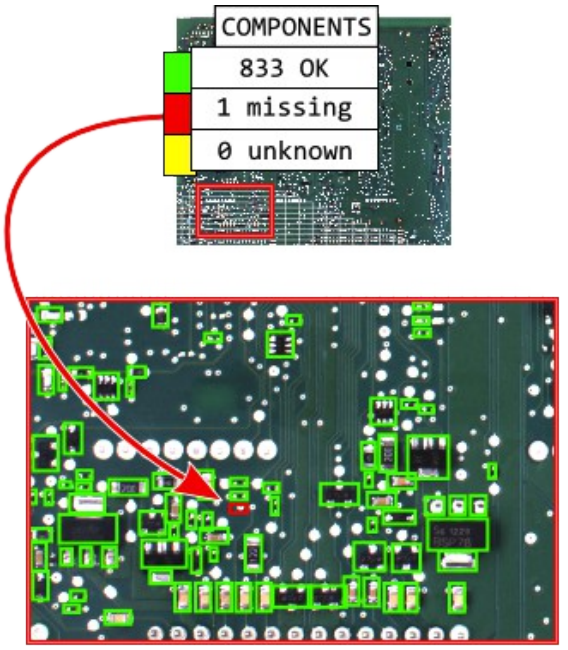
- Aplicações – robótica / carros autônomos:



# Visão computacional

- Aplicações – Controle de qualidade:



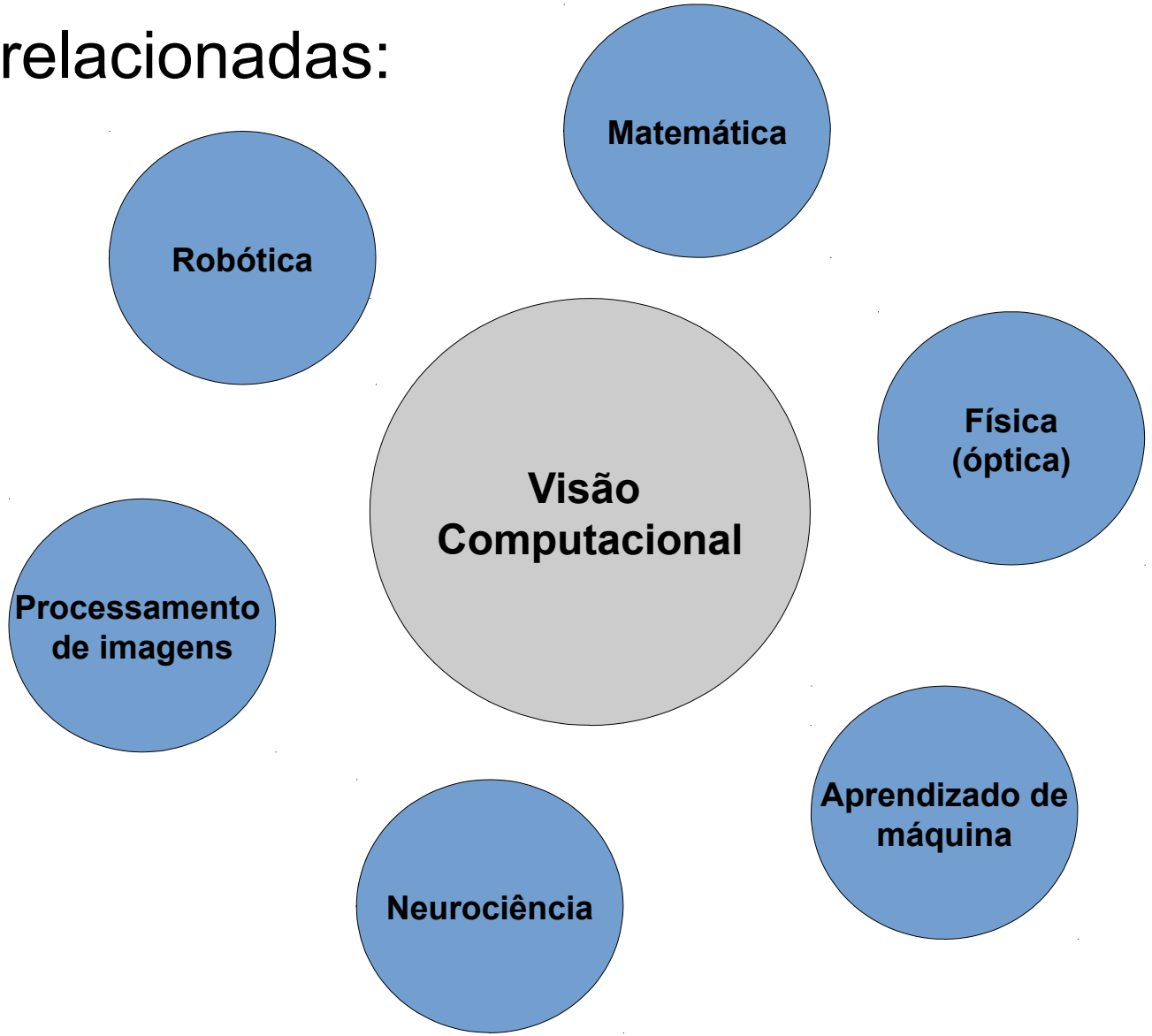



# Visão computacional

- Aplicações – entretenimento:


# Visão computacional

- Áreas relacionadas:



# Visão computacional

- Visão computacional x Computação gráfica

Cria um modelo computacional a partir da cena retratada na imagem

Cria uma imagem a partir de um modelo computacional

Visão computacional

Computação gráfica

**P(x,y)={(0,0),(0,10),(10,10),(10,0)}**
**RGB={0,0,255}**

# Visão computacional

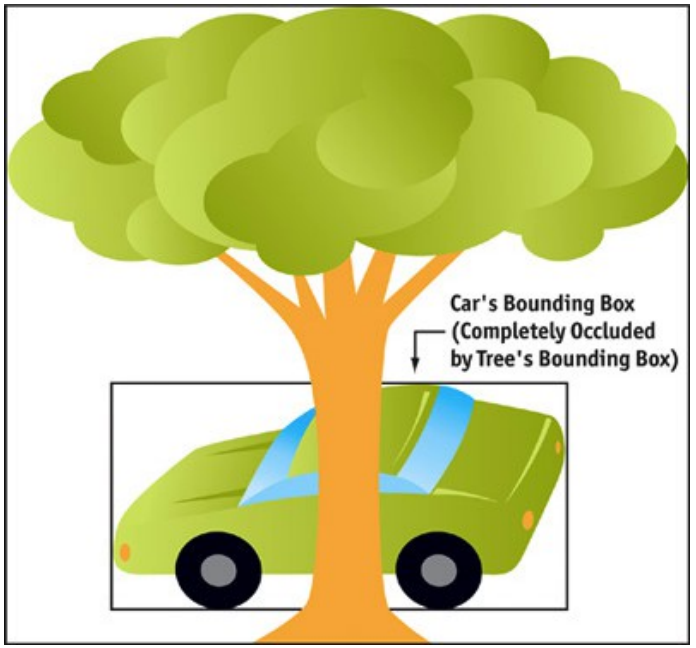
- **Como?**
  - Através de propriedades que podem ser extraídas das imagens
  - Inferir algo sobre a imagem a partir de suas propriedades e de um conhecimento prévio

# Desafios

- Oclusão
- Iluminação
- Imagem parcial
- Classes abrangentes
- Variações de um objeto
- Lacuna semântica

# Desafios

- Oclusão




# Desafios

- Iluminação

# Desafios

- Imagem parcial

?

# Desafios

- Imagem parcial

# Desafios

- Classes abrangentes

# Desafios

- Variações de uma classe

# Desafios

- Lacuna semântica

):

# Desafios

- Lacuna semântica

# Desafios

- Lacuna semântica

# Desafios

- Lacuna semântica

# Desafios

- Lacuna semântica

):

# Desafios

- Lacuna semântica

Visão computacional (VC**):** tentativa de replicar...

# Desafios

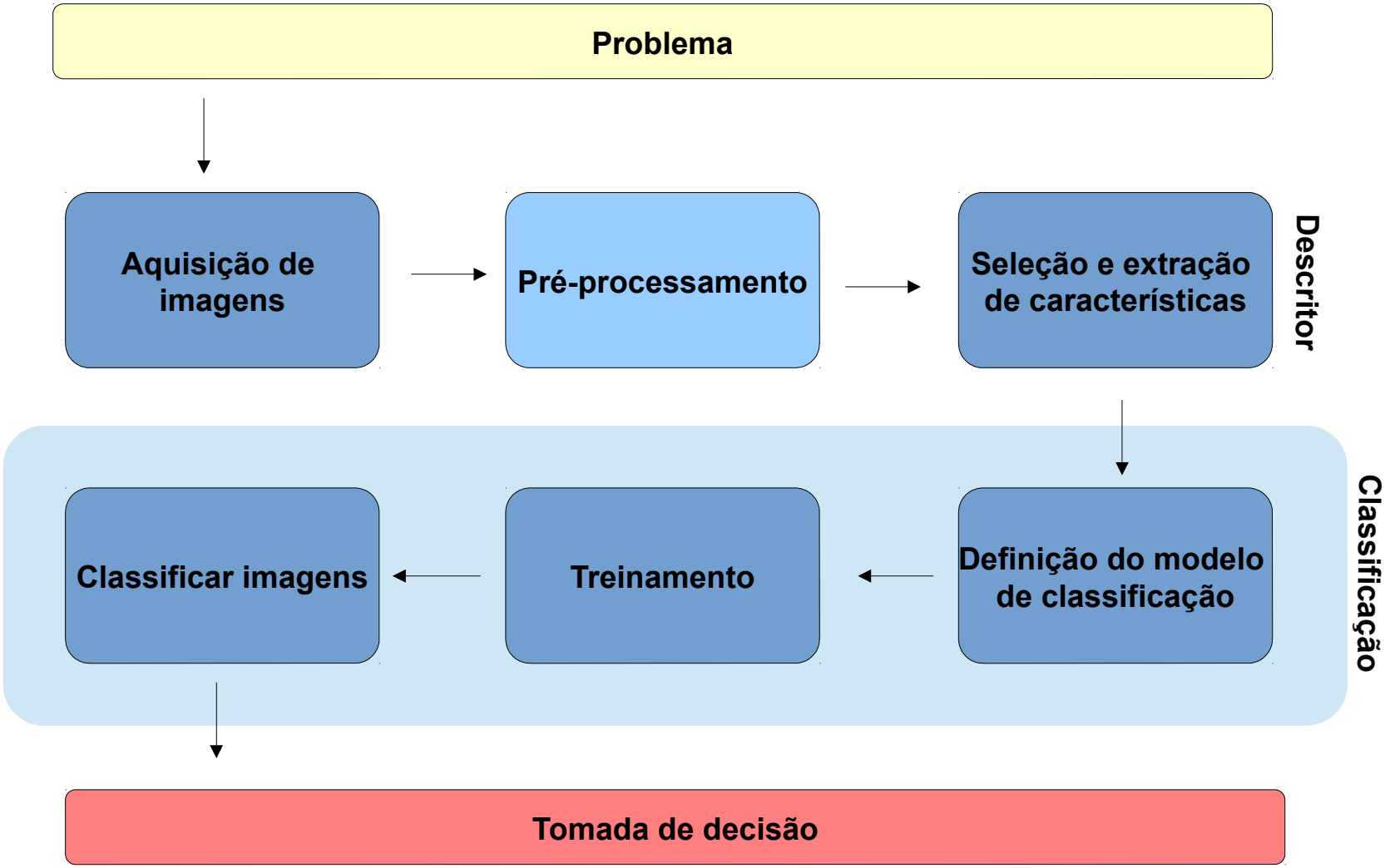
- Lacuna semântica
  - **Ausência** d e **relação** entre a informação visual extraída da **imagem** (baixo nível) e a **interpretação** que o mesmo dado tem para uma pessoa em uma determinada situação (contexto - alto nível);
  - Diferença entre duas descrições de um mesmo objeto a partir de diferentes representações;
  - A interpretação depende do **contexto**

# Desafios

- Lacuna semântica

# Visão computacional

- Processo:



# Aquisição da imagem

- **Imagens** ou **vídeos** digitais
- Trata a imagem como uma **matriz de pixels**

- **Considerar**:
  - Boa resolução
  - Compactação sem perdas
  - Controle de condições do ambiente (iluminação, posição da câmera, etc)

# Pré-processamento

- Etapa **opcional**
- Preparar dados para **auxiliar próximas etapas**
- Depende da tarefa e qualidade das amostras

- **Exemplos**:
  - Filtros
  - Segmentação

# Pré-processamento

- **Filtros**
  - Utilizados para:
    - **Aprimorar** imagens: **remover ruídos**, **redimensionar**, aumentar **contraste**, etc.
    - **Extrair informações** de imagens: textura, bordas, pontos salientes
    - Detectar **padrões**: template matching

Com ruído

Filtro gaussiano

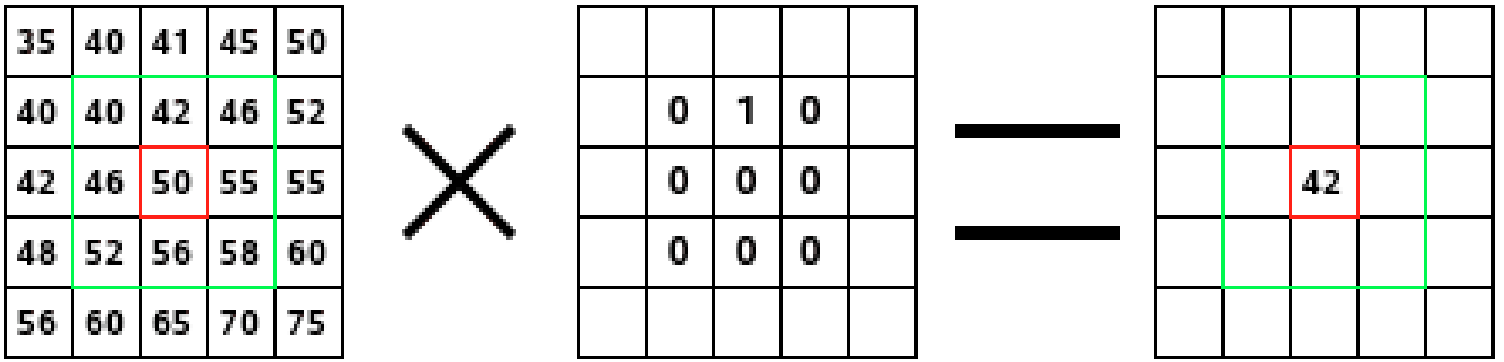
Filtro de média

sobel

# Pré-processamento

- **Filtros**
  - Cálculo de uma **função** aplicada, **pixel a pixe**l, onde o novo valor de um pixel depende do seu **valor original** e de seus **vizinhos**;

  - Convolução:




40x0 + 42x1 + 46x0 + 46x0 + 50x0 + 55x0 + 52x0 + 56x0 + 58x0 = **42**

# Pré-processamento

- **Filtro de média**

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 9 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

→

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

→

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

# Pré-processamento

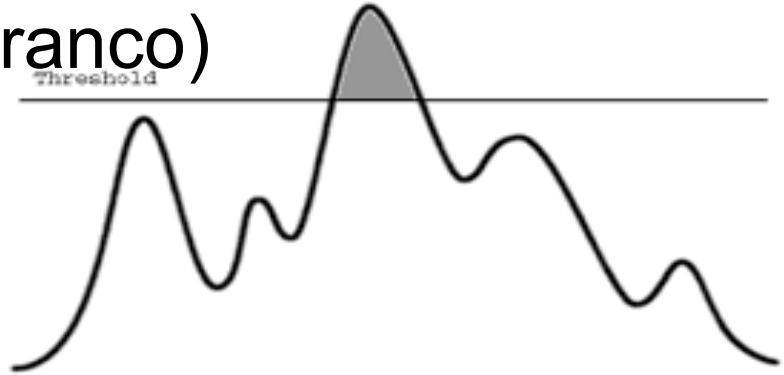
- **Segmentação**
  - **Dividir** a imagem em **regiões** (conjunto de pixels)
  - Regiões adjacentes possuem **diferenças** significativas de valor
  - **Simplificar** a imagem
  - Facilitar a análise
  - Métodos: Limiarização, watershed, Graph cuts, etc.

# Pré-processamento

- **Limiarização**
  - A partir do histograma de intensidade de cor da imagem, define-se um **ponto de corte** (ex. 100)

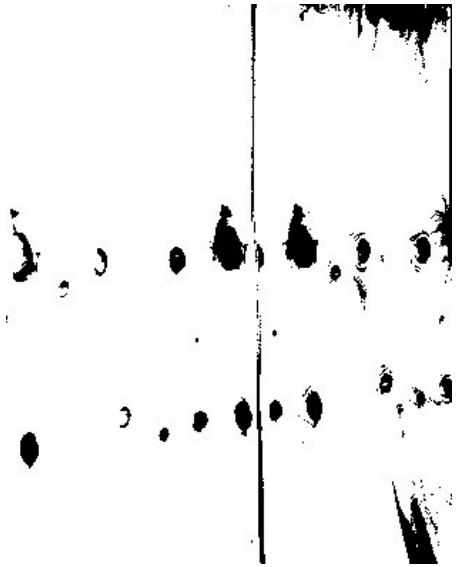
  - Valores maiores = 255
  - Valores menores = 0

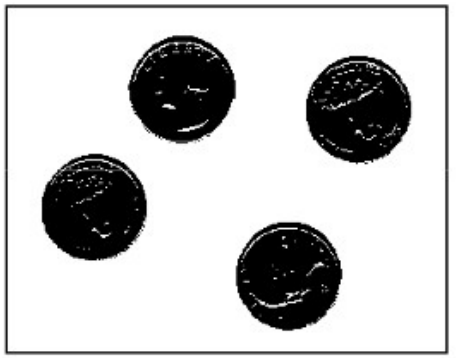
  - **Entrada**: imagem em níveis de cinza
  - **Saída**: imagem binária (preto e branco)

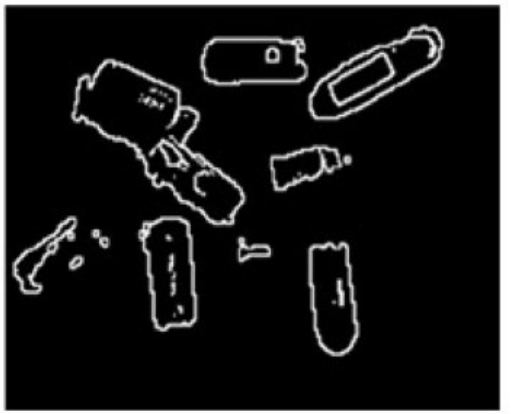


# Pré-processamento

- **Limiarização**



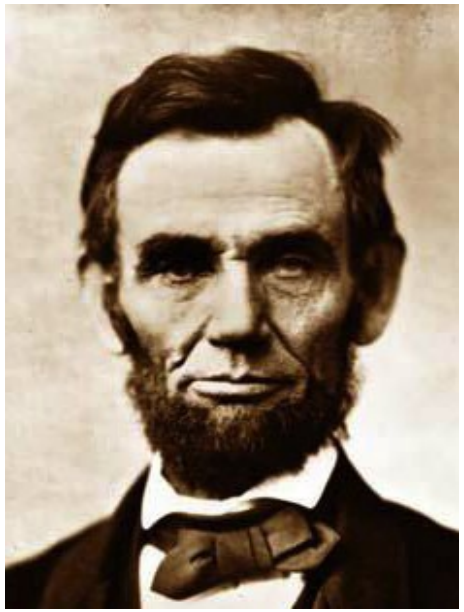



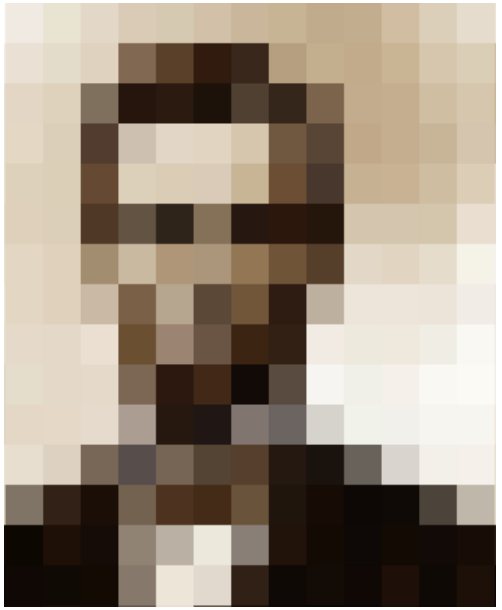
# Extração de características

- **Mapear** a imagem em um **vetor de características**
- Métodos **descritores**;
- Tipos de descritores de imagens:
  - **Locais**: pontos de interesse (SIFT, SURF)
  - **Globais**: imagem como um todo
    - Cor (BIC, JAC, histograma de cor)
    - Textura (SMS, LBP)
    - Forma (Fractal, Fourier)

- **Não existe** uma representação **única** ou **melhor** = depende da aplicação

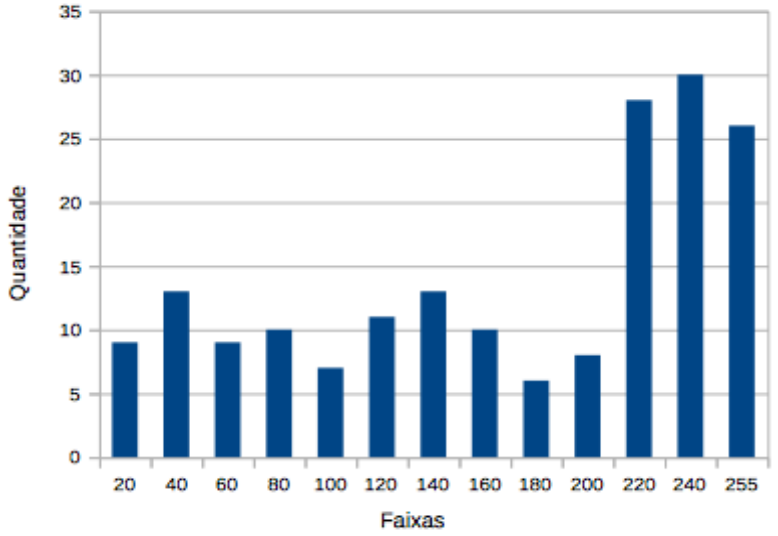
# Extração de características





| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 243 | 239 | 240 | 225 | 206 | 185 | 188 | 218 | 211 | 206 | 216 | 225 |
| 242 | 239 | 218 | 110 | 67 | 31 | 34 | 152 | 213 | 206 | 208 | 221 |
| 243 | 242 | 123 | 58 | 94 | 82 | 132 | 77 | 108 | 208 | 208 | 215 |
| 235 | 217 | 115 | 212 | 243 | 236 | 247 | 139 | 91 | 209 | 208 | 211 |
| 233 | 208 | 131 | 222 | 219 | 226 | 196 | 114 | 74 | 208 | 213 | 214 |
| 232 | 217 | 131 | 116 | 77 | 150 | 69 | 56 | 52 | 201 | 228 | 223 |
| 232 | 232 | 182 | 186 | 184 | 179 | 159 | 123 | 93 | 232 | 235 | 235 |
| 232 | 236 | 201 | 154 | 216 | 133 | 129 | 81 | 175 | 252 | 241 | 240 |
| 235 | 238 | 230 | 128 | 172 | 138 | 65 | 63 | 234 | 249 | 241 | 245 |
| 237 | 236 | 247 | 143 | 59 | 78 | | 94 | 255 | 248 | 247 | 251 |
| 234 | 237 | 245 | 193 | 55 | 33 | 115 | 144 | 213 | 255 | 253 | 251 |
| 248 | 245 | 161 | 128 | 149 | 109 | 138 | 65 | 47 | 156 | 239 | 255 |
| 190 | 107 | 39 | 102 | 94 | 73 | 114 | 58 | | | 51 | 137 |
| 23 | 32 | 33 | 148 | 168 | 203 | 179 | 43 | 27 | 17 | 12 | 8 |
| 17 | 26 | 12 | 160 | 255 | 255 | 109 | 22 | 26 | 19 | 35 | 24 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 243 | 239 | 240 | 225 | 206 | 185 | 188 | 218 | 211 | 206 | 216 | 225 |
| 242 | 239 | 218 | 110 | 67 | 31 | 34 | 152 | 213 | 206 | 208 | 221 |
| 243 | 242 | 123 | 58 | 94 | 82 | 132 | 77 | 108 | 208 | 208 | 215 |
| 235 | 217 | 115 | 212 | 243 | 236 | 247 | 139 | 91 | 209 | 208 | 211 |
| 233 | 208 | 131 | 222 | 219 | 226 | 196 | 114 | 74 | 208 | 213 | 214 |
| 232 | 217 | 131 | 116 | 77 | 150 | 69 | 56 | 52 | 201 | 228 | 223 |
| 232 | 232 | 182 | 186 | 184 | 179 | 159 | 123 | 93 | 232 | 235 | 235 |
| 232 | 236 | 201 | 154 | 216 | 133 | 129 | 81 | 175 | 252 | 241 | 240 |
| 235 | 238 | 230 | 128 | 172 | 138 | 65 | 63 | 234 | 249 | 241 | 245 |
| 237 | 236 | 247 | 143 | 59 | 78 | 10 | 94 | 255 | 248 | 247 | 251 |
| 234 | 237 | 245 | 193 | 55 | 33 | 115 | 144 | 213 | 255 | 253 | 251 |
| 248 | 245 | 161 | 128 | 149 | 109 | 138 | 65 | 47 | 156 | 239 | 255 |
| 190 | 107 | 39 | 102 | 94 | 73 | 114 | 58 | 17 | 7 | 51 | 137 |
| 23 | 32 | 33 | 148 | 168 | 203 | 179 | 43 | 27 | 17 | 12 | 8 |
| 17 | 26 | 12 | 160 | 255 | 255 | 109 | 22 | 26 | 19 | 35 | 24 |




**V = [9, 13, 9, 10, 7, 11, 13, 10, 6, 8, 28, 30, 26]**

# Classificação

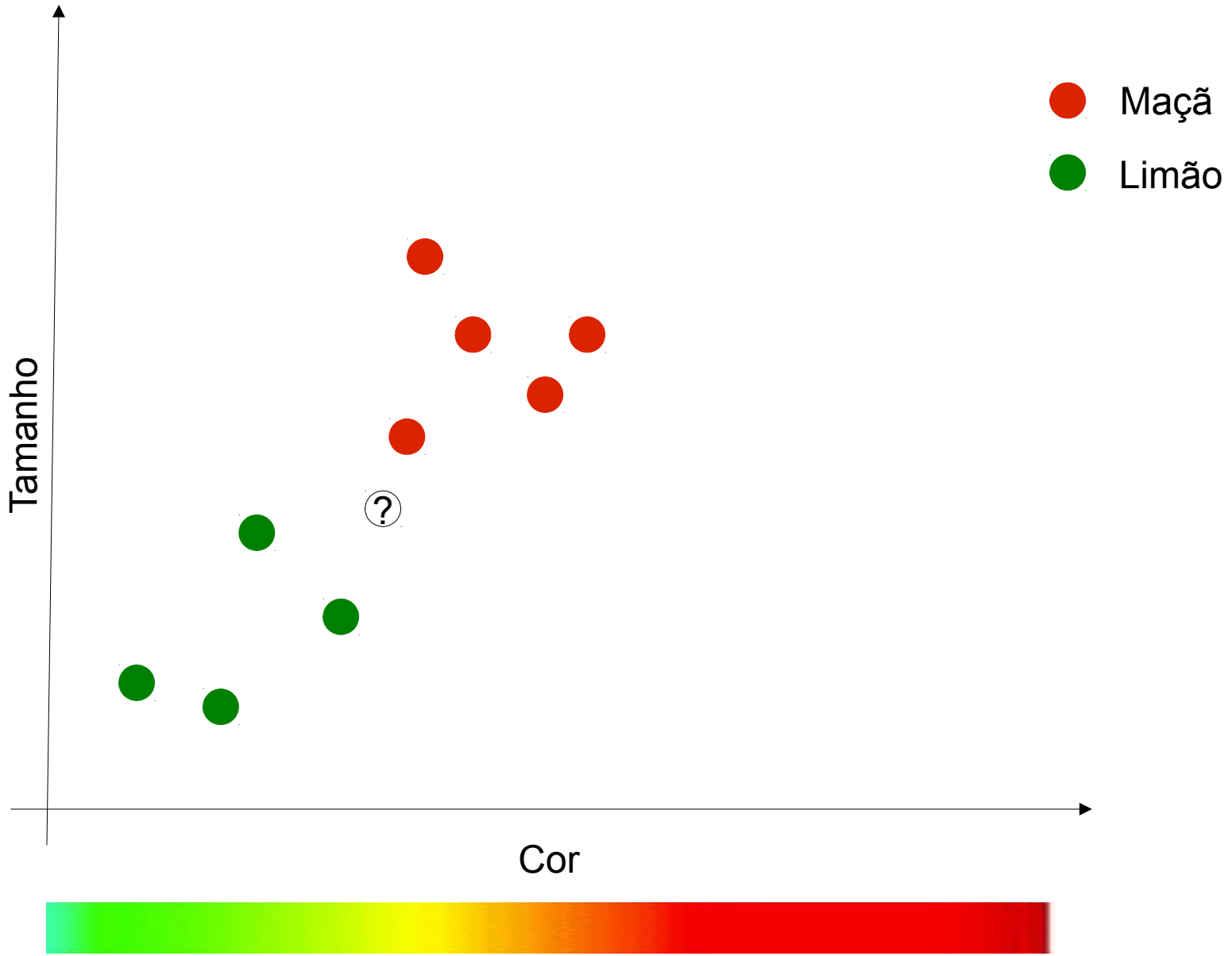
- Inferir a **qual classe** a imagem **pertence**
  - A qual delas é **mais similar**
- Aprendizado de máquina
- Métodos Classificadores
  - Entrada: vetor de características
  - Exemplos: kNN, SVM, redes neurais, OPF, etc.
  - Métodos supervisionados = treinamento
  - Função de distância
  - Identificar a qual classe a imagem é mais similar

# Tomada de decisão

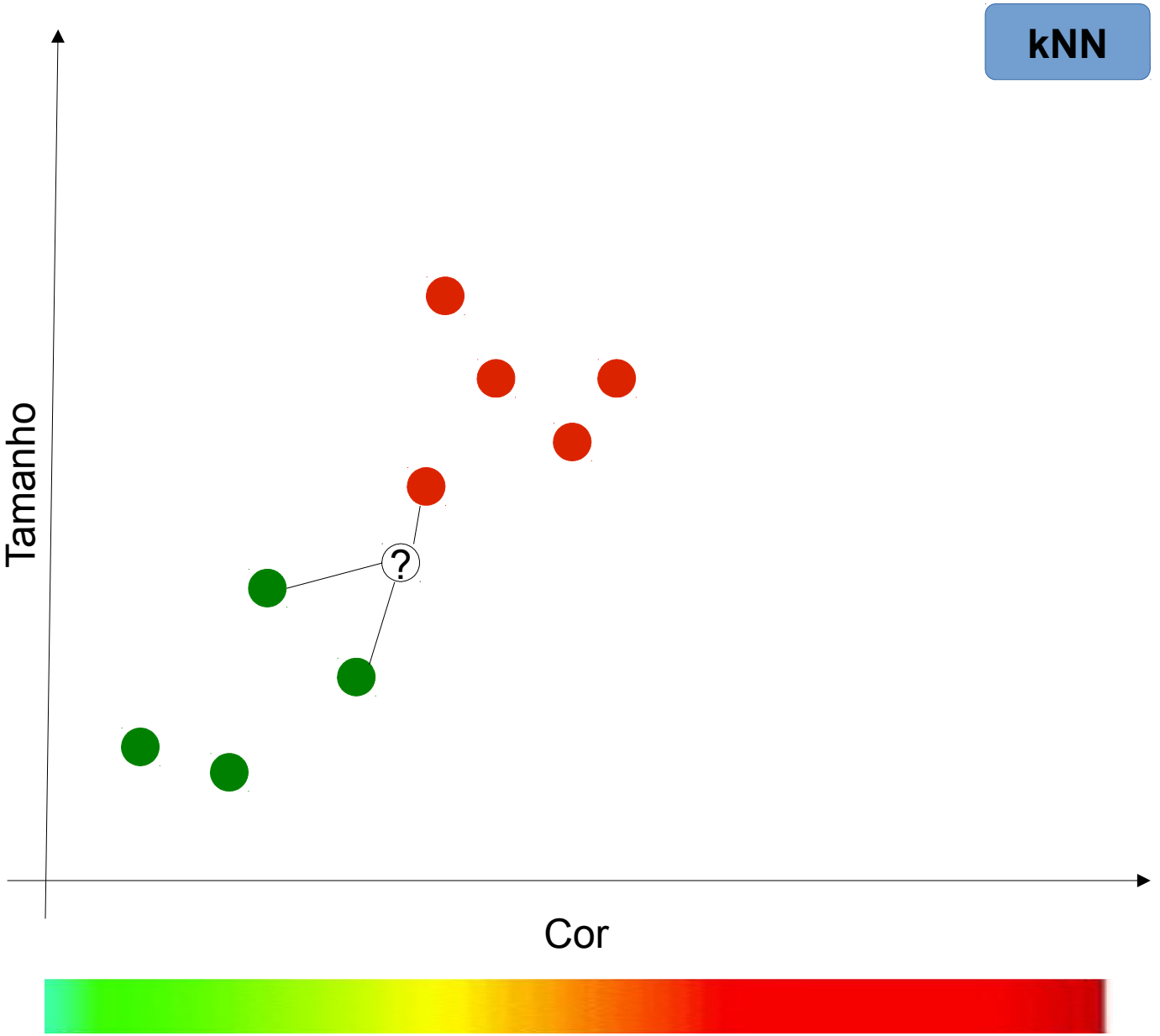
- A qual classe o objeto pertence?



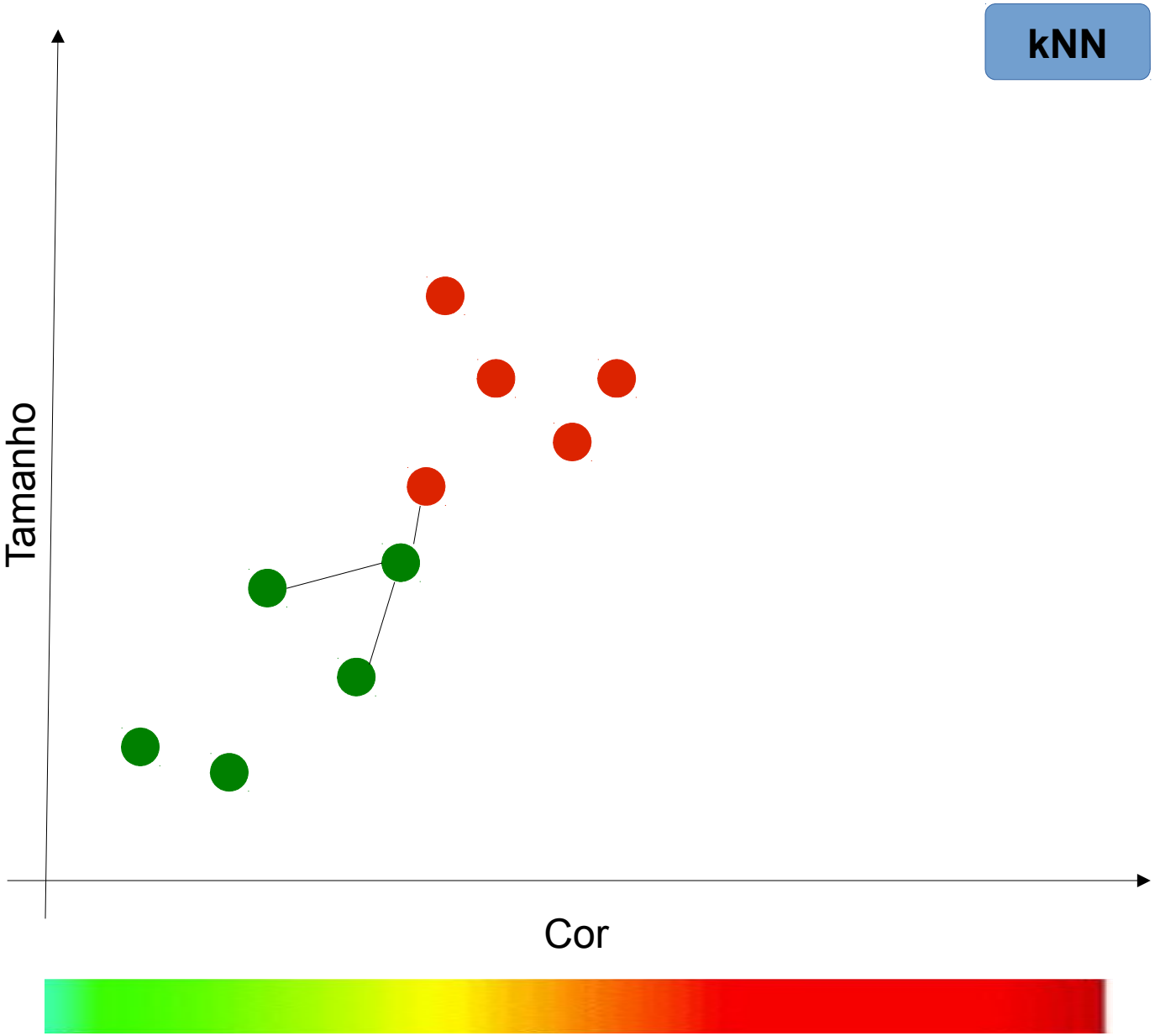
# Classificação



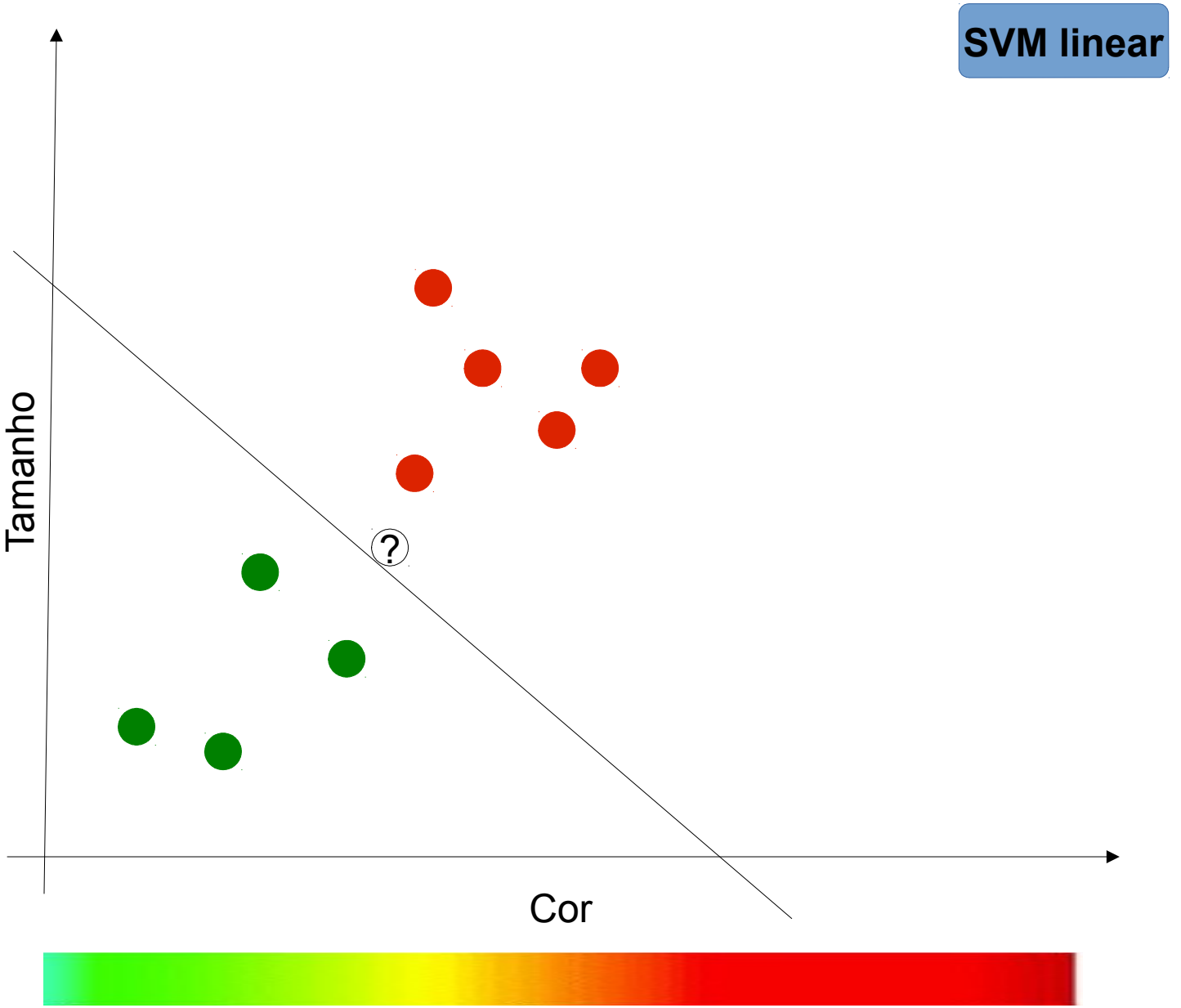
# Classificação



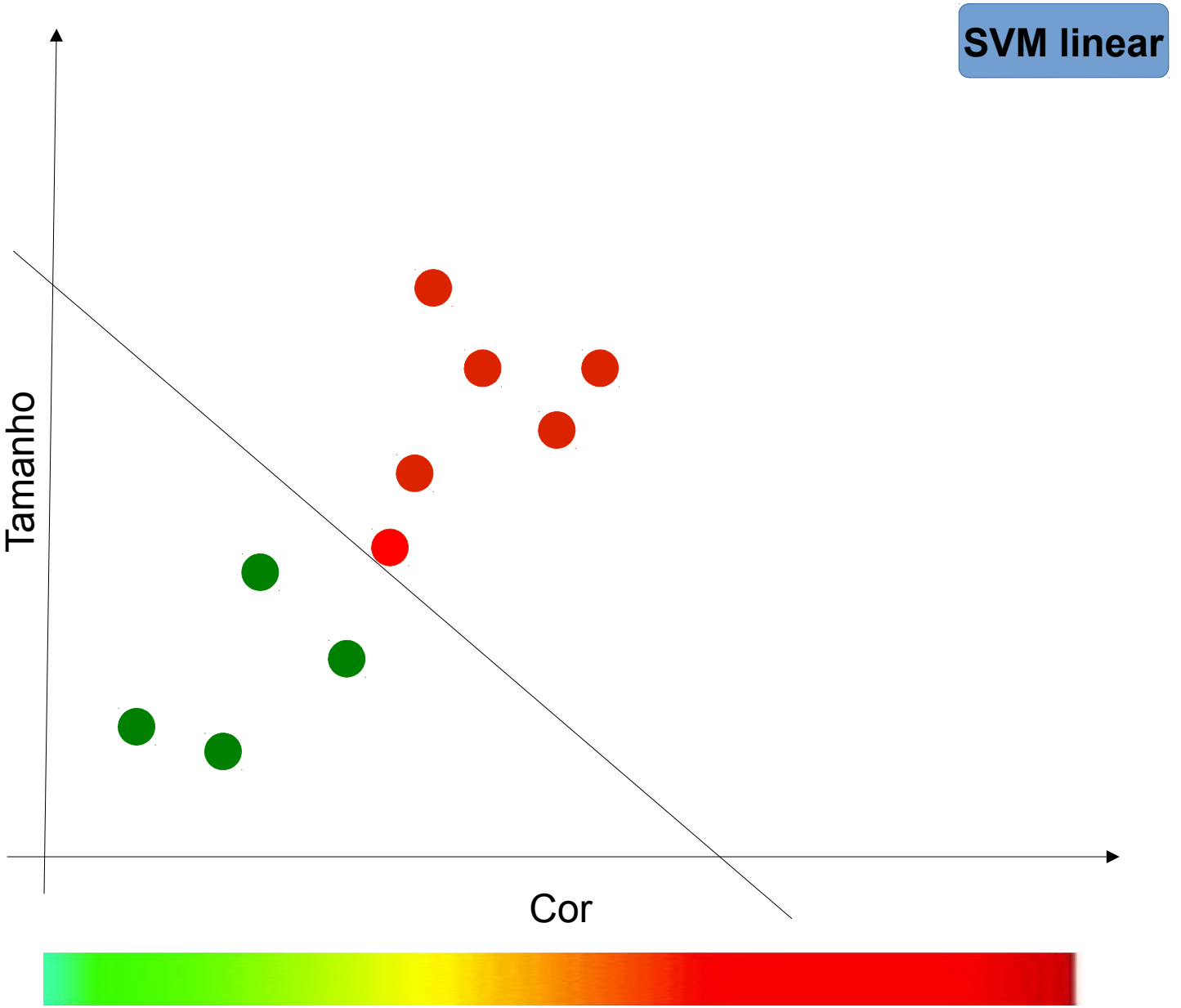
# Classificação



# Classificação



# Classificação



# Visão computacional

- Quais o melhor método?
- Depende:
  - Dados de entrada
  - O que pretende-se classificar
  - Informação que se tem sobre o conjunto de dados (categorias, etc)
  - Quantidade de classes
  - Qual característica destaca o objeto a ser identificado dos demais

# Visão computacional

- **Bibliotecas**:
  - VlFeat
  - OpenCV
  - http://www.computervisiononline.com/software

# Visão computacional

**Exemplo OpenCV (opencv/samples/c/smiledetect.cpp):**

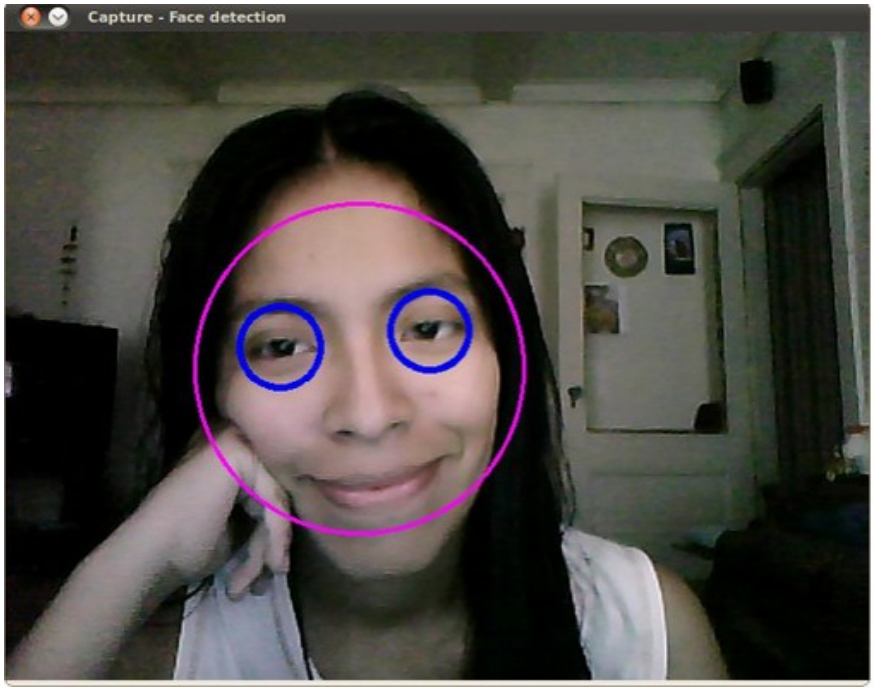
```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;
void detectAndDisplay( Mat frame );

String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
CascadeClassifier face_cascade, eyes_cascade;
string window_name = "Capture - Face detection";
RNG rng(12345);

int main( int argc, const char** argv ) {
    Mat img;
    //Carrega classificadores
    if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Erro\n"); return -1; };
    if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--(!)Erro\n"); return -1; };

    img = imread("imagem.jpg"); //carrega imagem
    if( !img.empty() )
    { detectAndDisplay( img ); }
    else
    { printf("Erro ao carregar imagem!"); break; }
    return 0;
}
```

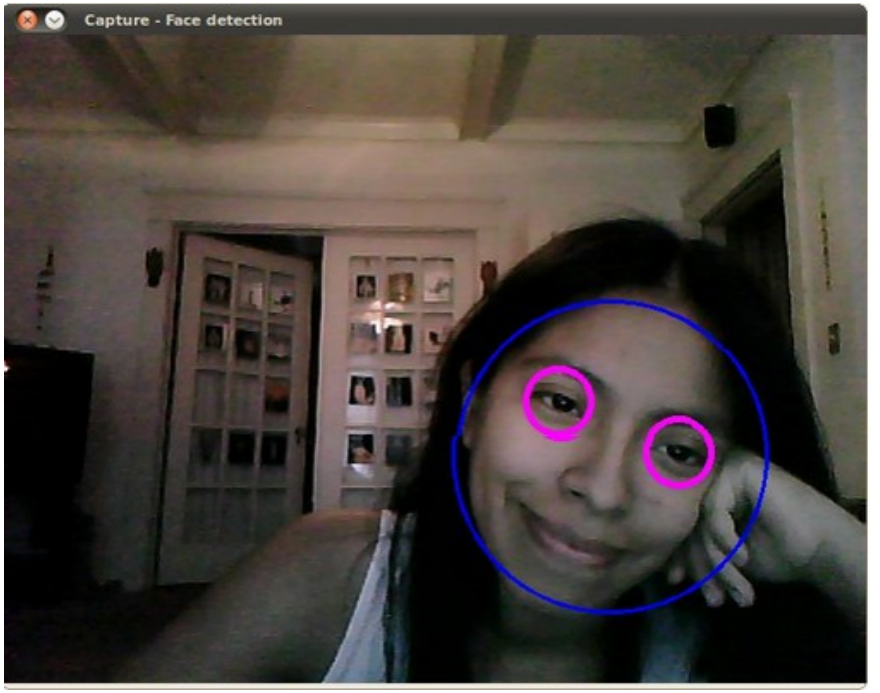
# Visão computacional

```
void detectAndDisplay( Mat img ){
    std::vector<Rect> faces;
    Mat img_gray;

    cvtColor( img, img_gray, CV_BGR2GRAY );
    equalizeHist( img_gray, img_gray );
    // Detecção da face
    face_cascade.detectMultiScale( img_gray, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

    for( size_t i = 0; i < faces.size(); i++ ) {
        Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
        ellipse( img, center, Size( faces[i].width*0.5,
                 faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );

        Mat faceROI = img_gray( faces[i] );
        std::vector<Rect> eyes;
        //Em cada face, detectar os olhos
        eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE, Size(30, 30) );

        for( size_t j = 0; j < eyes.size(); j++ ) {
            Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
                          faces[i].y + eyes[j].y + eyes[j].height*0.5 );
            int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
            circle( img, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
        }
    }
    imshow( window_name, img );
}
```

# Visão computacional

**Resultado:**




**Documentação**: http://docs.opencv.org/