

# Documentação Técnica do Modelo de Visão Computacional para Detecção de Frutas Estragadas

---

## 1. Introdução

O objetivo deste documento é apresentar a documentação técnica completa do modelo de visão computacional desenvolvido para a classificação de frutas em dois estados: Fresh (fresca) e Rotten (estragada). O modelo utiliza pré-processamento de imagem, redução de dimensionalidade via PCA e um classificador SVM.

## 2. Dataset e Pré-processamento

As imagens foram normalizadas para faixa de 0 a 1, convertidas para float32 e achatadas. Para reduzir a dimensionalidade, utilizou-se PCA. O conjunto foi dividido em treino, validação e teste.

## 3. Arquitetura do Modelo

A arquitetura do modelo foi construída como um pipeline completo de visão computacional. O fluxo é composto pelas seguintes etapas principais:

### 1. Organização e Limpeza do Dataset

Antes do pipeline de aprendizado, o código realiza:

- Download automático do dataset via KaggleHub
- Limpeza de versões antigas do KaggleHub para redução de espaço
- Buscas automáticas das pastas com nome *Fresh* e *Rotten*
- Eliminação de duplicatas estruturais
- Reorganização das imagens em estrutura padronizada:
  - /train/fresh, /train/rotten
  - /val/fresh, /val/rotten
  - /test/fresh, /test/Rotten
- Split com separação 70-15-15

Essa etapa garante consistência e evita vazamento de dados entre conjuntos de treino, validação e teste.

### 2. Pré-processamento das Imagens

Cada imagem passa por um pipeline de limpeza e padronização composto por:

#### 2.1. Redimensionamento e Normalização

- Converte para RGB
- Ajusta para 128x128 pixels
- Normaliza para faixa [0, 1] (float32)

## 2.2. Segmentação da Fruta

O método `segment_fruit()` aplica:

- Conversão para HSV
- Criação de máscara baseada em saturação e valor
- Fechamento e abertura morfológica para remover ruídos
- Aplicação da máscara na imagem
- Resizing + normalização final

## 2.3. CLAHE

Equalização adaptativa de iluminação, melhorando contraste em imagens mais escuras. Desativada pois o aumento do contraste estava inferindo no modelo detectar mais manchas falsas, resultando em mais Falsos Positivos (classificar como estragadas as que eram frescas) durante os testes.

## 2.4. Data Augmentation

Para aumentar a variabilidade do dataset de treino, foi adicionada uma função de aumento de imagens chamada `augment_image()`. Essa função realiza transformações aleatórias sobre cada imagem, incluindo:

- Rotação: ângulo aleatório entre  $-20^\circ$  e  $20^\circ$ ;
- Flip horizontal e vertical: cada um com 50% de chance;
- Ajuste de brilho: incremento aleatório entre -30 e +30;
- Normalização final: converte novamente para float32 com valores entre 0 e 1.

Essas operações simulam variações reais de captura (como diferentes ângulos, iluminação e orientação da fruta), tornando o modelo mais robusto e generalizável.

## 3. Extração de Features

O modelo se baseia em técnicas clássicas de extração manual de atributos:

### 3.1. Histogramas HSV (Cor)

- Histograma de 32 bins para cada canal H, S e V
- Concatenação formando um vetor de 96 dimensões
- Normalização para soma 1

*Captura variações de cor associadas ao apodrecimento da fruta (escurecimento, perda de saturação, esverdeamento, etc).*

### 3.2. Textura via Local Binary Pattern (LBP)

- Conversão para escala de cinza;
- Cálculo de LBP uniforme com  $(P=8, R=1)$
- Geração de histograma normalizado de padrões locais

*Essa técnica é eficiente para detectar:*

- *Rugosidade da casca*
- *Manchas*
- *Áreas mofadas*
- *Texturas irregulares*

#### 4. Carregamento Otimizado com Paralelização (Multithreading)

A extração de features é acelerada usando:

- `ThreadPoolExecutor(max_workers=8)`
- Processamento paralelo de cada imagem
- Retorno dos vetores de features já prontos para treino

*Isso reduz drasticamente o tempo total.*

#### 5. Redução de Dimensionalidade com PCA

Após o dataset ser carregado e convertido em features numéricas, aplica-se:

***PCA (n\_components = 50, whiten=True)***

Objetivos:

- Remover redundância entre atributos
- Reduzir ruído
- Tornar o SVM mais estável
- Aumentar separabilidade linear entre classes
- Acelerar o treinamento

*O whiten=True garante que todos os componentes finais tenham variância normalizada, beneficiando o classificador linear.*

#### 6. Classificação com SVM (LinearSVC)

O classificador final é:

***LinearSVC(C=1.0, max\_iter=5000)***

Escolha técnica:

- LinearSVC funciona muito bem com features de histograma e LBP
- Tem custo computacional baixo
- Funciona bem com PCA whitened
- É robusto a ruídos e dados levemente correlacionados

O parâmetro `C=1.0` controla a margem entre classes.

O `max_iter=5000` evita problemas de convergência.

#### 7. Pipeline Unificado (sklearn Pipeline)

Tudo é integrado no pipeline:

```
Pipeline([
    ("pca", PCA(...)),
    ("svm", LinearSVC(...))
])
```

**Vantagens:**

- Permite reproducibilidade
- Treina e testa sempre com a mesma sequência de transformações

## 8. Avaliações

Modelo produz:

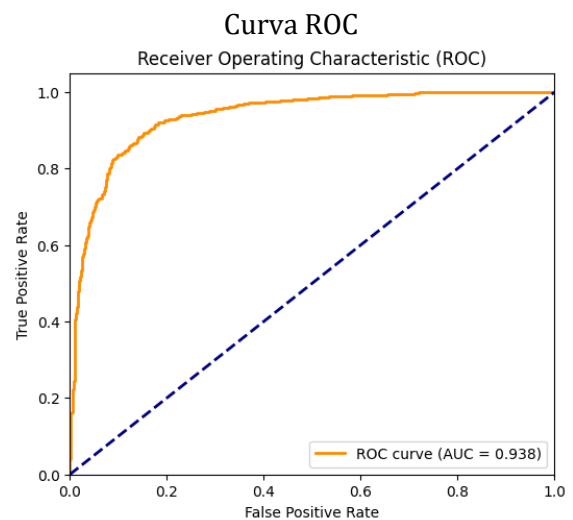
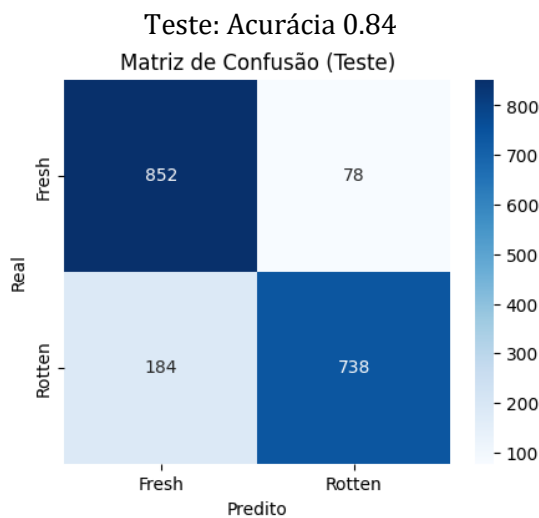
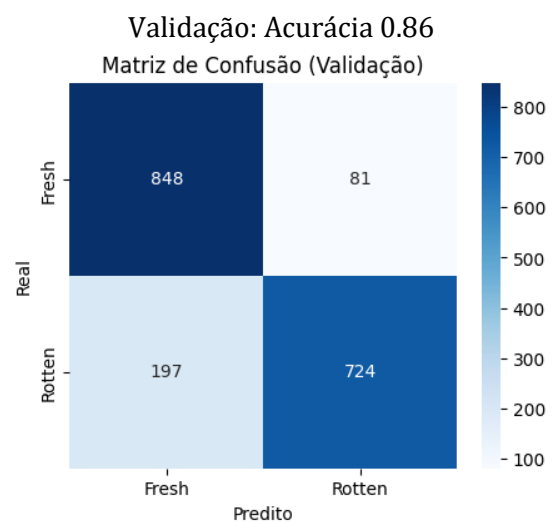
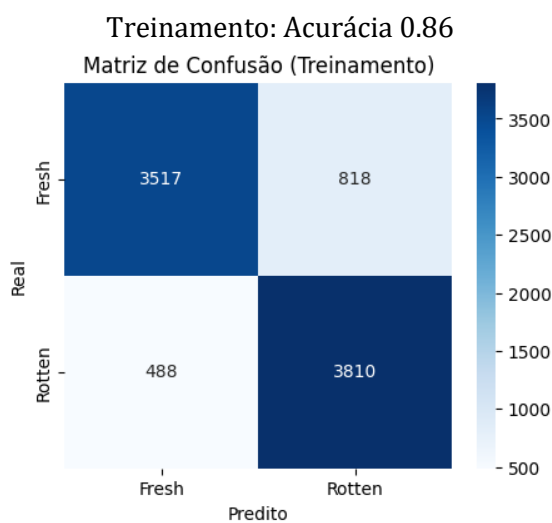
- Classification Report: precisão, recall, F1
- Matriz de confusão com seaborn
- Curva ROC e AUC
- Avaliações separadas para treino, validação e teste

A curva ROC utiliza decision\_function do SVM, gerando AUC robusto.

Essa etapa garante consistência e evita vazamento de dados entre os conjuntos de treino, validação e teste, que era um problema no último modelo apresentado em sala.

## 4. Métricas de Desempenho

Resultados obtidos nos arquivos de relatório:



*Fresh: recall ~0.81–0.92, precisão ~0.81–0.88*

*Rotten: recall ~0.79–0.89, precisão ~0.82–0.90*

## 5. Interpretação das Métricas

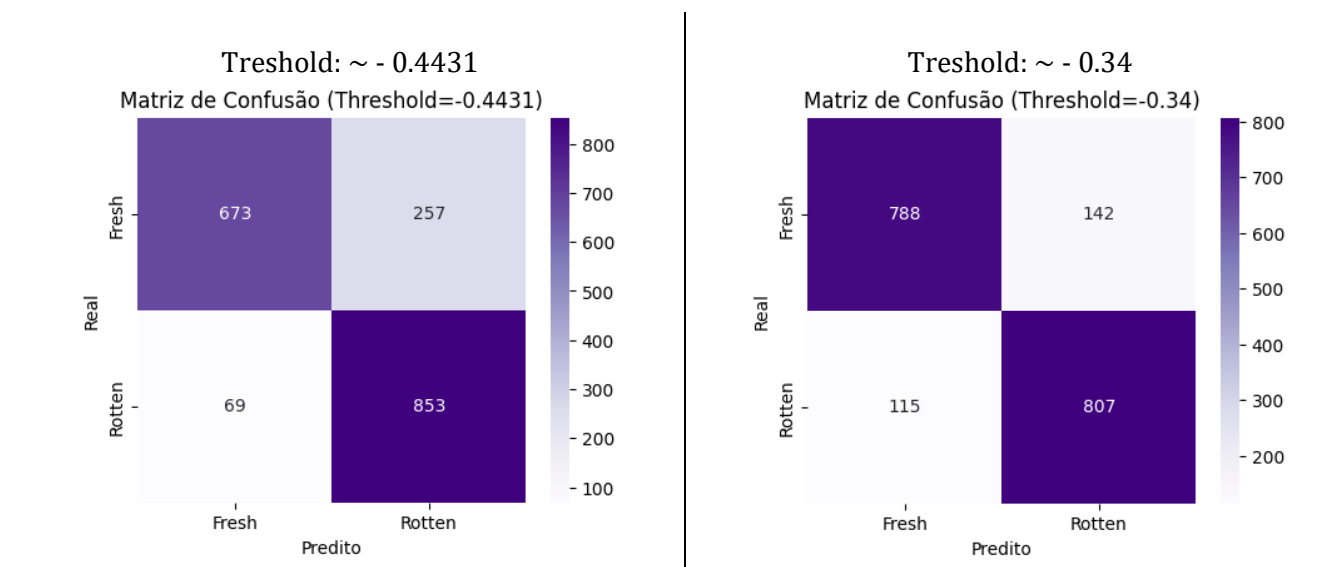
As métricas mostram desempenho consistente entre treino, validação e teste, sem indícios de overfitting, ou seja, do modelo estar decorando os dados da etapa de treinamento.

Aparenta ter um leve underfitting, pois o modelo poderia capturar melhor a classe Rotten, que tem um recall menor do que de Fresh.

Com um AUC por volta de 0.9, o modelo demonstra uma boa separação entre Rotten e Fresh. Isso mostra que se pegarmos duas imagens aleatórias, uma Rotten e outra Fresh, o modelo irá atribuir um score maior ao Rotten cerca de ~90% das vezes.

Na Curva ROC, podemos ver que ela sobe muito próxima do eixo Y, o que mostra que a True Positive Rate aumenta muito enquanto a False Positive Rate ainda é baixa. Isso mostra que o modelo consegue identificar de maneira correta uma grande quantidade de amostras positivas logo no início, sem cometer muitos erros de falso positivo. À medida que a curva avança, ela começa a ir em direção ao canto superior direito do gráfico, pois conforme reduzimos o limiar, o modelo passa a classificar mais amostras como positivas, o que consequentemente aumenta a taxa de falsos positivos. Ainda assim, a curva mantém um formato com inclinação sustentada, o que sugere que o modelo tem boa capacidade de diferenciar as classes em diferentes faixas limiares.

Após rodar um trecho de código para achar um threshold melhor do que o padrão 0, chegamos ao threshold de  $\sim -0.4431$ . Ao utilizar esse limiar, percebemos o aumento de FP em relação aos FN, assim como a diminuição de TN e aumento de TP. Como o objetivo do modelo não é priorizar a classificação nem de Fresh nem de Rotten, decidimos aumentar um pouco o limiar para  $\sim -0.34$ , o que equilibrou um pouco mais as classificações. Essa mudança não afetou de maneira expressiva a taxa de TP, mas melhorou de maneira considerável a taxa de TN, o que o grupo considerou como uma boa mudança no limiar de classificação.



## 6. Matriz de Confusão e Viés

FP e FN estão equilibrados. O modelo não favorece excessivamente nenhuma das classes e apresenta bom balanço entre precisão e recall.

## 7. Conclusão

O modelo apresenta desempenho sólido, com boa capacidade de generalização.

Foram feitas tentativas de aplicar um modelo mais robusto para a classificação das imagens, com CNNs, porém o Google Colab não suportou todas as operações que estavam sendo feitas.

## 8. Limites do Modelo

Ele depende exclusivamente de sinais visuais. Alguns alimentos deteriorados não apresentam alterações externas no início, o que impede o modelo de identificar o problema.

Outro ponto é a variabilidade das condições do usuário: iluminação ruim, sombras, câmera fraca ou ângulos incomuns podem gerar erros. Tentamos mitigar esse ponto com `data_augmentation`, aplicando aleatoriamente nas imagens de treino rotação (ângulo aleatório entre  $-20^\circ$  e  $20^\circ$ ), flip horizontal e vertical (cada um com 50% de chance), e ajuste de brilho (incremento aleatório entre -30 e +30).

Utilizar um dataset pequeno ou com poucas frutas estragadas para o treinamento do modelo pode gerar sobreajuste e reduzir a capacidade de generalização.

Alterações químicas e microbiológicas não são detectadas pelo modelo visual — nossa solução é auxiliar, não substituir inspeção humana adequada.

**Ressalva:** grandes dificuldades na classificação correta de morangos. Identificamos como possível causa o fato do morango tipicamente, em grande parte dos casos, manter sua cor vermelha mesmo quando podre, o que dificulta o modelo identificar a diferença entre Rotten e Fresh dessa fruta.

## 9. Próximos passos

- Aumentar o dataset com fotos de frutas cortadas, classificadas em Rotten e Fresh, a fim de expandir a capacidade de classificação do modelo. O modelo é capaz de classificar imagens de frutas cortadas, porém utilizar esses padrões no treino aumentariam a capacidade do modelo.
- Identificar frutas de maneira distintas dentro de imagens com mais de uma fruta, a fim de dizer separadamente quais estão frescas e quais estão podres.

## 10. Links

Link Google Colab

- Google Colab: [Modelo Fresh-Rotten | Google Colab](#)

Link Dataset Kaggle: Augmented Dataset for Fruits (Rotten/Fresh)

- Kaggle: [Augmented Dataset for Fruits \(Rotten/Fresh\)](#)

Link GitHub (repositório do projeto)

- GitHub: [Repositório no GitHub](#)