

**UNIVERSIDADE PRESBITERIANA MACKENZIE**  
**CURSO DE TECNOLOGIA EM CIÊNCIA DE DADOS**

**ANDERSON APARECIDO DA SILVA ALVES**  
**ANDRÉIA DOMINGOS DOS SANTOS**  
**GERSON SOARES RODRIGUES**  
**SAMUEL BONFIM DA SILVA**

**RA:**

**10347602**

**10288503**

**10423804**

**10423509**

**APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS PARA**  
**RECONHECIMENTO DE FRUTAS**

**SÃO PAULO**

**2024**

**ANDERSON APARECIDO DA SILVA ALVES**  
**ANDRÉIA DOMINGOS DOS SANTOS**  
**GERSON SOARES RODRIGUES**  
**SAMUEL BONFIM DA SILVA**

**APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS PARA**  
**RECONHECIMENTO DE FRUTAS**

Análise de dados fornecidos pela plataforma Zenodo

Trabalho acadêmico para conclusão da disciplina de Projeto Aplicado II do Curso de Tecnologia em Ciência de Dados pela Universidade Presbiteriana Mackenzie.

Professor: Felipe Albino dos Santos

**SÃO PAULO**

**2024**

A001

Silva, Anderson Aparecido da Silva.

Dos Santos, Andréia Domingos.

Rodrigues, Gerson Soares.

Da Silva, Samuel Bonfim.

Análise de Frutas por meio de Redes Neurais Convolucionais (CNN) / Anderson Aparecido da Silva Alves; Andréia Domingos dos Santos; Gerson Soares Rodrigues; Samuel Bonfim da Silva - 2024. 38 f.: il.; 30 cm.

Trabalho Acadêmico (Projeto Aplicado II) – Universidade Presbiteriana Mackenzie,  
Presbiteriana Mackenzie, São Paulo, 2024.

## SUMÁRIO

1. INTRODUÇÃO .....	1
2. GLOSSÁRIO .....	1
3. OBJETIVO E METAS.....	2
4. METODOLOGIA.....	2
5. APRESENTAÇÃO DOS DADOS.....	3
6. ANÁLISE EXPLORATÓRIA.....	3
7. ANÁLISE EXPLORATÓRIA DOS DADOS DO PROJETO .....	3
8. TREINAMENTO DO MODELO DE TESTE .....	9
9. AVALIAÇÃO DO MODELO DE TESTE.....	15
10. REALIZAÇÃO DE TRANSFORMAÇÃO E AVALIAÇÃO DO UTILIZANDO COMO BASE DE TESTE A BASE DE DADOS TRATADAS QUE SERÃO EMPREGADAS NO PROJETO.....	23
11. CONJUNTO DE TESTE UTILIZANDO MÉTRICAS ADICIONAIS COM GERAÇÃO DE MATRIZ DE CONFUSÃO E RELATÓRIO DE CLASSIFICAÇÃO .....	25
12. MODELO DE NEGÓCIOS .....	31
13. EXPECTATIVA PARA AS PROXIMAS ENTREGAS .....	32
14. CONCLUSÃO .....	32
15. LINK PARA O GITHUB .....	33
16. REFERÊNCIA BIBLIOGRÁFICA .....	33

## 1. INTRODUÇÃO

O reconhecimento automático de frutas é um desafio global que afeta predominantemente o setor de comércio e logística, especialmente em supermercados e mercados de alimentos. Essa dificuldade é causada por fatores como a grande semelhança visual entre diferentes tipos de frutas e as variações ambientais, como iluminação e posição. Com o aumento da automação e da necessidade de precisão nas operações comerciais, identificar e classificar corretamente os produtos tornou-se crucial. Desde o avanço das técnicas de visão computacional, a precisão no reconhecimento de imagens tem melhorado, mas ainda existem desafios significativos. Para enfrentar esse problema, diversas soluções foram desenvolvidas, incluindo o uso de redes neurais convolucionais (CNN), que permitem o reconhecimento de frutas com alta precisão, mesmo em condições desafiadoras. O projeto utiliza redes neurais convolucionais (CNN) para o reconhecimento automatizado de frutas, com o objetivo de melhorar a eficiência em supermercados e empresas de logística que requerem precisão na classificação de produtos alimentícios. O modelo usa a arquitetura ResNet50 com Transfer Learning, aplicada a um dataset de 44.406 imagens, permitindo a identificação de frutas com alta precisão.

## 2. GLOSSÁRIO

As imagens utilizadas neste projeto foram disponibilizadas pela plataforma Zenodo, (nos formatos JPG, JPEG e PNG) que estão armazenados no GitHub do projeto e analisadas nas próximas etapas;

Para este projeto, as bibliotecas utilizadas nesse trabalho até o momento têm como os seguintes objetivos:

- **os**: Manipula funcionalidades do sistema operacional, como caminhos de arquivos e diretórios.
- **torch**: Principal biblioteca para computação em tensor e aprendizado profundo em PyTorch.
- **re**: Realiza operações de expressão regular para manipulação e busca de strings.
- **cv2**: Biblioteca OpenCV para processamento de imagens e visão computacional.
- **shutil**: Facilita operações de alto nível em arquivos e coleções de arquivos, como copiar e mover.
- **random**: Gera números aleatórios e realiza operações relacionadas à aleatoriedade.

- **PIL (Image):** Manipula imagens, como abrir, criar e salvar arquivos de imagem.
- **numpy:** Biblioteca para computação numérica, facilitando operações em arrays e matrizes.
- **torch.nn:** Fornece classes e funções para construir redes neurais em PyTorch.
- **seaborn:** Facilita a visualização de dados estatísticos com gráficos mais informativos.
- **pandas:** Manipula e analisa dados estruturados em DataFrames.
- **torch.optim:** Implementa algoritmos de otimização para treinar modelos em PyTorch.
- **matplotlib.pyplot:** Cria gráficos e visualizações de dados.
- **torchvision:** Oferece conjuntos de dados, transformações e modelos pré-treinados para tarefas de visão computacional.
- **DataLoader:** Carrega conjuntos de dados em lotes para treinamento de modelos em PyTorch.
- **torchvision.models:** Fornece modelos pré-treinados para tarefas de visão computacional.
- **sklearn.metrics:** Avalia o desempenho do modelo por meio de métricas como precisão, recall e F1-score.
- **collections.defaultdict:** Cria dicionários que fornecem um valor padrão para chaves inexistentes.
- **sklearn.model\_selection:** Divide conjuntos de dados em treino e teste, e realiza validação cruzada.
- **confusion\_matrix, classification\_report:** Avaliam e relatam a precisão de um modelo de classificação.

### 3. OBJETIVO E METAS

**Objetivo:** Criar uma solução analítica para o reconhecimento de frutas que diminua o tempo e aumente a precisão na identificação de produtos, contribuindo para a automação dos processos no setor comercial.

**Metas:** Alcançar alta acurácia (acima de 90%) no reconhecimento de diferentes frutas, utilizando métricas de avaliação que englobam acurácia, precisão, recall e F1-score.

### 4. METODOLOGIA

**Coleta de Dados:** Dataset de frutas obtido através da plataforma Zenodo, com 44.406 imagens de diferentes tipos de frutas, variando em condições de iluminação e posição.

**Pré-processamento:** Normalização das imagens para tamanhos e escalas padronizados, separação dos dados em treino (70%), validação (15%) e teste (15%).

**Modelagem com Transfer Learning:** Utilização da ResNet50 com camadas de ajuste para as classes específicas de frutas, permitindo um modelo de classificação com generalização e precisão.

## 5. APRESENTAÇÃO DOS DADOS

Os dados que serão apresentados nesse trabalho foram adquiridos da seguinte plataforma:

<https://zenodo.org/records/1310165>

O Dataset é composto por 44.406 imagens de frutas, coletadas em um período de 6 meses. As imagens foram feitas em um ambiente de laboratório em diferentes cenários mencionadas no artigo ao fim desse trabalho.

## 6. ANÁLISE EXPLORATÓRIA

Nesta etapa, realizamos uma análise detalhada do conjunto de dados de frutas para entender suas características e assegurar que o modelo tenha uma base sólida para o treinamento. As principais atividades realizadas foram:

- **Identificação de Categorias e Distribuição de Imagens:** Avaliamos a quantidade de imagens por categoria de fruta, analisando a distribuição dos dados para garantir a representatividade de cada tipo de fruta no modelo.
- **Descrição Estatística dos Dados:** Calculamos estatísticas como média, desvio padrão e outros valores descritivos para verificar a consistência e variabilidade entre as categorias. Isso incluiu a análise de variações de iluminação e posição para identificar potenciais desafios.
- **Visualização de Imagens:** Realizamos a plotagem de imagens aleatórias do conjunto para inspeção visual e confirmação da uniformidade. Essa análise visual ajudou a identificar se as imagens estavam organizadas e categorizadas de maneira adequada para o modelo, além de destacar as condições de iluminação e ângulos de captura, que podem impactar na precisão do reconhecimento.

Essas análises exploratórias foram fundamentais para entender as características do conjunto de dados e preparar uma base sólida para as próximas etapas do projeto, garantindo que o modelo consiga generalizar e performar bem em condições variadas.

## 7. ANÁLISE EXPLORATÓRIA DOS DADOS DO PROJETO

Iniciamos nosso projeto utilizando como compilador dos dados o Jupyter Anaconda e posteriormente, para que a leitura fosse feita através do acesso pelo GitHub, o Google Colaboratory, comumente chamado de Google Colab. Os scripts foram desenvolvidos em Python, utilizando bibliotecas como os, torch, re, cv2, shutil, Random, PIL, numpy, pandas, seaborn, matplotlib, collections e sklearn. O pré-processamento incluiu a exploração dos diretórios e arquivos, verificação da quantidade de imagens para trabalho, a análise de distribuição por tipo e quantidade por frutas presentes na base de dados, separação dos dados em que as imagens serão separadas em conjuntos (tipos) de frutas. Estes passos poderão ser visualizados através das instruções contidas nas imagens e linhas de códigos a seguir:

### Importação das Bibliotecas utilizadas no Projeto

```
import os
import torch
import re
import cv2
import shutil
import random
from PIL import Image
import numpy as np
import torch.nn as nn
import seaborn as sns
import pandas as pd
from PIL import Image
import torch.optim as optim
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torchvision import models
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score, recall_score
from torchvision.models import ResNet50_Weights
from collections import defaultdict
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

### Diretório utilizado para a adição das imagens

```
dir_principal =
"C:/Users/samue/OneDrive/Documentos/PROJETO_APLICADO"
dataset = os.path.join(dir_principal, "Fruit Data _Base")
```

### Explorando Diretórios e Arquivos com os.walk()

```
for root, dirs, files in os.walk(dir_principal):
    print(f'Diretório: {root}') # Exibe o diretório atual
```



```

    for filename in files:
        print(f'    Encontrado arquivo: {filename}')    # Exibe cada
arquivo encontrado

```

## Quantidade de imagens para trabalho

```

# Dicionário para armazenar a contagem de imagens por fruta
fruit_counts = defaultdict(int)

# Contador para total geral de imagens
total_images = 0

image_extensions = ('.png', '.jpg', '.jpeg')

# Função para normalizar o nome da fruta
def normalize_fruit_name(name):
    # Extrai a parte principal do nome da fruta
    match = re.match(r'^(apple).*', name, re.IGNORECASE)
    return match.group(1).lower() if match else name.lower()

# Percorrendo todas as subpastas e arquivos
for root, dirs, files in os.walk(dataset):
    for filename in files:
        if filename.lower().endswith(image_extensions):    # Verifica
se o arquivo é uma imagem PNG
            # Normaliza o nome da fruta a partir da subpasta
            fruit_name =
normalize_fruit_name(os.path.basename(root))
            # Incrementa a contagem para a fruta correspondente
            fruit_counts[fruit_name] += 1
            # Incrementa o contador total de imagens
            total_images += 1

# Exibindo os resultados
print("Total de imagens por tipo de fruta:")
for fruit, count in fruit_counts.items():
    print(f'Total de imagens para {fruit}: {count}')

# Exibindo o total geral de imagens
print(f"\nTotal geral de imagens: {total_images}")

```

E, como saída, teremos o total de imagens por tipo de frutas:

```
➡ Total de imagens por tipo de fruta:  
Total de imagens para apple: 5024  
Total de imagens para banana: 3027  
Total de imagens para carambola: 2080  
Total de imagens para guava: 4008  
Total de imagens para kiwi: 4173  
Total de imagens para mango: 4154  
Total de imagens para muskmelon: 2078  
Total de imagens para orange: 3012  
Total de imagens para peach: 2629  
Total de imagens para pear: 3012  
Total de imagens para persimmon: 2072  
Total de imagens para pitaya: 2501  
Total de imagens para plum: 2298  
Total de imagens para pomegranate: 2167  
Total de imagens para tomatoes: 2171  
  
Total geral de imagens: 44406
```

Figura 1: Total de imagens geral e separas por tipos de frutas.

### Análise de Distribuição

```
fruit_df = pd.DataFrame(list(fruit_counts.items()),  
columns=['Fruit', 'Count'])  
# Configurando o estilo do gráfico  
sns.set(style="whitegrid")  
  
# Criando um gráfico de barras  
plt.figure(figsize=(12, 6))  
sns.barplot(x='Fruit', y='Count',  
data=fruit_df.sort_values('Count', ascending=False))  
plt.xticks(rotation=90) # Rotaciona os rótulos do eixo x para  
melhor legibilidade  
plt.title('Distribuição de Imagens por Tipo de Fruta')  
plt.xlabel('Tipo de Fruta')  
plt.ylabel('Número de Imagens')  
plt.show()
```

E, como saída, teremos o gráfico de Distribuição de Imagens por Tipo de Fruta

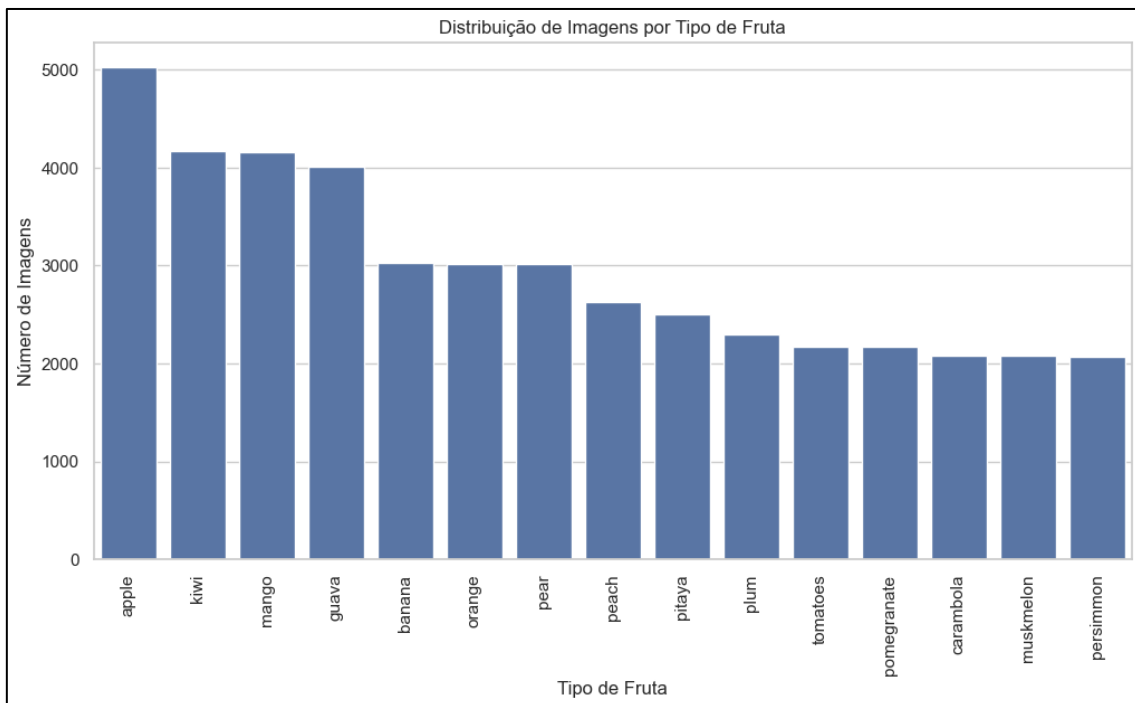


Figura 2: Gráfico de Distribuição de Imagens por Tipo de Fruta.

Logo após a distribuição das imagens por tipos de frutas, criamos um DataFrame a partir de um dicionário de contagens de frutas, em que, através dos comandos que serão informados a seguir, serão exibidas: as primeiras linhas, informações gerais e estatísticas descritivas do DataFrame:

```
# Criando um DataFrame a partir do dicionário de contagens de
# frutas
fruit_df = pd.DataFrame(fruit_counts.items(), columns=['Fruit',
'Count'])

# Exibindo as primeiras linhas do DataFrame
print(fruit_df.head())

# Exibindo informações gerais do DataFrame
print(fruit_df.info())

# Exibindo estatísticas descritivas
print(fruit_df.describe())
```

E, como saída, teremos as informações referentes à programação acima que poderemos visualizar a seguir:

```

⇒      Fruit  Count
0      apple  5024
1     banana  3027
2   carambola  2080
3      guava  4008
4       kiwi  4173
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    Fruit    15 non-null    object
1    Count    15 non-null    int64
dtypes: int64(1), object(1)
memory usage: 372.0+ bytes
None

              Count
count      15.000000
mean     2960.400000
std       949.286333
min      2072.000000
25%      2169.000000
50%      2629.000000
75%      3517.500000
max      5024.000000

```

Figura 3: Saída da programação acima com as informações de contagem da quantidade das 5 primeiras frutas, informações gerais e dados estatísticos do DataFrame.

## Plotagem dos Dados

```

# Função para mostrar imagens aleatórias
def show_random_images(fruit_name, dataset, num_images=3):
    # Caminho da pasta da fruta
    fruit_path = os.path.join(dataset, fruit_name)
    images = [img for img in os.listdir(fruit_path) if
img.lower().endswith(image_extensions)]

    # Seleciona aleatoriamente as imagens
    selected_images = random.sample(images, min(num_images,
len(images)))

    # Plotando as imagens
    plt.figure(figsize=(15, 5))
    for i, img_name in enumerate(selected_images):
        img_path = os.path.join(fruit_path, img_name)
        img = Image.open(img_path)
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'{fruit_name}: {img_name}')
    plt.show()

# Mostrando amostras para algumas frutas
for fruit in fruit_df['Fruit'].sample(n=3).values: # Seleciona
aleatoriamente 3 frutas
    show_random_images(fruit, dataset)

```

Através da função utilizada acima, selecionaremos aleatoriamente grupos (tipo) de frutas e destes são selecionadas 3 imagens por grupo, conforme ilustra a imagem a seguir:



Figura 4: Imagens selecionadas aleatoriamente de acordo com o grupo, em que foram escolhidos três grupos e destas três imagens por grupo.

### Separação dos Dados

```
def listar_subdiretorios(dir):
    return [os.path.join(dir, nome)
            for nome in os.listdir(dir)
            if os.path.isdir(os.path.join(dir, nome))]

data_dir = listar_subdiretorios(dir_principal)
```

No comando acima, a função *listar\_subdiretorios* retorna uma lista de subdiretórios dentro de um diretório fornecido. O código *data\_dir = listar\_subdiretorios(dir\_principal)* armazena essa lista em *data\_dir*.

## 8. TREINAMENTO DO MODELO DE TESTE

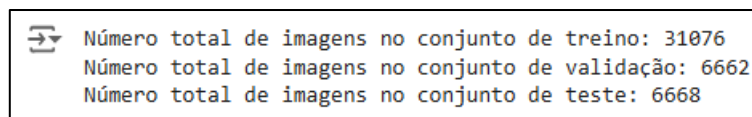
Decidimos durante o processo de pesquisa utilizar o método Transfer Learning da ResNet50 que apresentou excelentes resultado quando aplicados aos dados com grande diversidade de categorias.

No comando a seguir, o código definirá três diretórios (*train\_dir*, *val\_dir*, *test\_dir*), uma função para contar imagens e exibir o número de imagens em cada diretório (treino, validação e teste).

```
base_dest_dir =
"C:/Users/samue/OneDrive/Documentos/PROJETO_APLICADO/new_dataset_fr
utas"
train_dir, val_dir, test_dir = [os.path.join(base_dest_dir, x) for
x in ['train', 'val', 'test']]
# Função para contar imagens em um diretório
def contar_imagens(dir):
    return sum([len(files) for _, _, files in os.walk(dir)])

# Contar e exibir as imagens em cada conjunto
for dir_name, dir_path in zip(["treino", "validação", "teste"],
[train_dir, val_dir, test_dir]):
    print(f"Número total de imagens no conjunto de {dir_name}:
{contar_imagens(dir_path)}")
```

E como saída, teremos o número total de imagens dos conjuntos *treino*, *validação* e *teste* conforme a ilustrado na imagem abaixo:



```
➞ Número total de imagens no conjunto de treino: 31076
Número total de imagens no conjunto de validação: 6662
Número total de imagens no conjunto de teste: 6668
```

Figura 5: Número total de imagens dos conjuntos de treino, validação e teste.

As linhas de códigos irão contar o número de imagens em cada conjunto (treino, validação e teste), calcular o total de imagens e a porcentagem de cada conjunto, exibindo esses resultados.

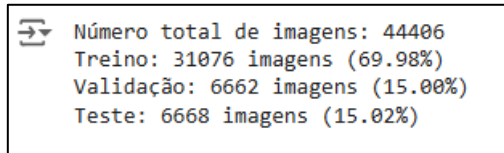
```
# Contar as imagens em cada conjunto
num_train = contar_imagens(train_dir)
num_val = contar_imagens(val_dir)
num_test = contar_imagens(test_dir)

# Número total de imagens
total_imagens = num_train + num_val + num_test

# Calcular porcentagens
porc_train = (num_train / total_imagens) * 100
porc_val = (num_val / total_imagens) * 100
porc_test = (num_test / total_imagens) * 100
```

```
# Exibir os resultados
print(f"Número total de imagens: {total_imagens}")
print(f"Treino: {num_train} imagens ({porc_train:.2f}%)")
print(f"Validação: {num_val} imagens ({porc_val:.2f}%)")
print(f"Teste: {num_test} imagens ({porc_test:.2f}%)")
```

E, como saída, teremos o número total de imagens, e o total de valores e porcentagens de treino, validação e teste, conforme ilustrado na figura abaixo:



```
⇒ Número total de imagens: 44406
Treino: 31076 imagens (69.98%)
Validação: 6662 imagens (15.00%)
Teste: 6668 imagens (15.02%)
```

Figura 6: Número total de imagens, treino, validação e teste.

## Transformação dos dados de treino

Usamos como métricas de transformação de dados a documentação apresentada pelo Pytorch, que apresentou uma grande eficiência aos nossos dados. Decidimos durante o processo de pesquisa utilizar o método.

O código a seguir irá preparar as imagens para o treinamento e validação de um modelo. Serão definidas as transformações como redimensionamento, flip horizontal (no treino), conversão para tensor e normalização baseada na ImageNet. Em seguida, serão carregadas as imagens das pastas de treino e validação usando ImageFolder, o qual aplica essas transformações e organiza os dados em lotes com DataLoaders para otimizar o treinamento. Por fim, serão determinados o número de classes no conjunto de treino para uso posterior no modelo.

```
# Definir as transformações para os dados
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(), # Aumento de dados
        transforms.ToTensor(), # Converter para tensor
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225]) # Normalizar com média e desvio padrão de ImageNet
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ])
```

```

    ])
}

# Carregar os dados
image_datasets = {x:
datasets.ImageFolder(root=f'{base_dest_dir}/{x}',
transform=data_transforms[x])
                    for x in ['train', 'val']}

# DataLoaders para carregar os dados em lotes
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32,
shuffle=True, num_workers=4)
                for x in ['train', 'val']}

# Número de classes no dataset
num_classes = len(image_datasets['train'].classes)

```

## Método de Transfer Learning da Arquitetura ResNet50

Utilizaremos a arquitetura ResNet50 pré-treinada com pesos do ImageNet para aproveitar o aprendizado anterior em imagens que ocorre através do congelamento das camadas convolucionais para evitar seu ajuste, substitui a camada final para corresponder ao número de classes do conjunto de dados e transfere o modelo para a GPU (se disponível) para acelerar o processamento. Estas informações poderão ser observadas através das linhas de comandos descritas a seguir:

```

# Carregar o modelo ResNet50 com os pesos mais recentes do ImageNet
model = models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# Congelar as camadas convolucionais
for param in model.parameters():
    param.requires_grad = False

# Substituir a última camada para o número correto de classes
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, num_classes)

# Enviar o modelo para a GPU
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model = model.to(device)

# Verificando a GPU
print(torch.cuda.is_available())

```



E logo após a instrução descrita dentro do print, como confirmação da GPU, O *True* deverá ser apresentado na saída, conforme a imagem abaixo:

```
[ ] # Verificando a GPU
    print(torch.cuda.is_available())

True
```

Figura 7: Confirmação da GPU através da visualização do *True* na saída.

O código a seguir irá definir a função de perda como *CrossEntropyLoss* para classificação e fará uso do otimizador *Adam* para ajustar apenas a última camada do modelo com uma taxa de aprendizado de 0.001.

```
# Definir a função de perda
criterion = nn.CrossEntropyLoss()

# Definir o otimizador (apenas para os parâmetros da última camada)
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
```

Nos comandos a seguir utilizaremos como:

**Configuração e Modo de Treino** a função *train\_model* para definir o modelo em modo de treino e preparar para as épocas de treinamento.

No **Loop de Épocas** a função treinada por um número de épocas (*num\_epochs*). Em cada época, reinicia contadores de perda (*running\_loss*) e acertos (*running\_corrects*) para acompanhar o desempenho.

No **Processamento de Batches** Em cada lote, os dados serão transferidos para a GPU (se disponível); os gradientes serão zerados para evitar acúmulos; será realizada uma passagem direta (*forward pass*) para se obter as previsões e o cálculo da perda por meio da utilização do comando *criterion* que executa o backpropagation (*loss.backward()*) para calcular os gradientes, e o otimizador (*optimizer.step()*) atualizará os pesos da última camada.

O **Cálculo de Estatísticas** ocorrerá após cada lote, acumulando a perda total e o número de acertos. Ao final de cada época, calculará a perda média (*epoch\_loss*) e a acurácia (*epoch\_acc*) e exibirá esses resultados.

Por fim, na **Execução**, a função *train\_model* será chamada para treinar o modelo com os parâmetros definidos, monitorando o desempenho. Essas instruções poderão ser visualizadas através das linhas de comando que serão descritas logo abaixo:

```
# Função para treinar o modelo
def train_model(model, criterion, optimizer, dataloaders, device,
num_epochs=10):
    model.train() # Colocar o modelo em modo de treino
    for epoch in range(num_epochs):
```

```

running_loss = 0.0
running_corrects = 0

for inputs, labels in dataloaders['train']:
    inputs = inputs.to(device)
    labels = labels.to(device)

    # Zerar os gradientes
    optimizer.zero_grad()

    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, labels)

    # Backward pass e otimização
    loss.backward()
    optimizer.step()

    # Estatísticas
    _, preds = torch.max(outputs, 1)
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == labels.data)

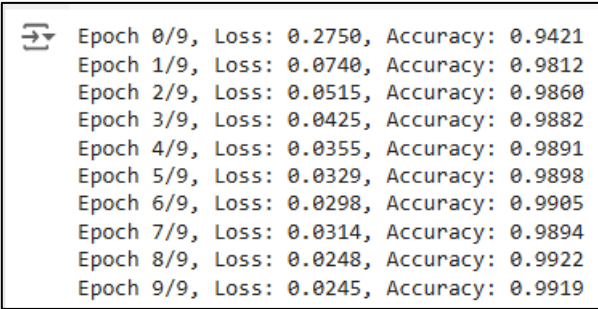
epoch_loss = running_loss /
len(dataloaders['train'].dataset)
epoch_acc = running_corrects.double() /
len(dataloaders['train'].dataset)

print(f'Epoch {epoch}/{num_epochs - 1}, Loss:
{epoch_loss:.4f}, Accuracy: {epoch_acc:.4f}')

# Treinar o modelo
train_model(model, criterion, optimizer, dataloaders, device,
num_epochs=10)

```

Como saída, a figura a seguir irá ilustrar a quantidade de épocas e seus valores de perdas e acurácia:



```

Epoch 0/9, Loss: 0.2750, Accuracy: 0.9421
Epoch 1/9, Loss: 0.0740, Accuracy: 0.9812
Epoch 2/9, Loss: 0.0515, Accuracy: 0.9860
Epoch 3/9, Loss: 0.0425, Accuracy: 0.9882
Epoch 4/9, Loss: 0.0355, Accuracy: 0.9891
Epoch 5/9, Loss: 0.0329, Accuracy: 0.9898
Epoch 6/9, Loss: 0.0298, Accuracy: 0.9905
Epoch 7/9, Loss: 0.0314, Accuracy: 0.9894
Epoch 8/9, Loss: 0.0248, Accuracy: 0.9922
Epoch 9/9, Loss: 0.0245, Accuracy: 0.9919

```

Figura 8: Quantidade de Épocas (Epoch) e suas respectivas Perdas (Loss) e Acurácia (Accuracy).

Implementado em PyTorch com Transfer Learning da ResNet50. O treinamento durou 8 horas, com ajuste fino das camadas finais para adaptação às classes de frutas.

## 9. AVALIAÇÃO DO MODELO DE TESTE

### Modelo em modo de avaliação

O código a seguir irá definir e executar uma função para avaliar o desempenho do modelo no conjunto de validação através de:

**Configuração do Modo de Avaliação:** A função `evaluate_model` colocará o modelo em modo de avaliação (`model.eval()`), desativando camadas como dropout, o que é importante para a avaliação consistente do modelo.

**Desativação do Cálculo de Gradientes:** Utilizando `torch.no_grad()` para desativar o cálculo de gradientes durante a avaliação, memória será economizada, melhorando assim a velocidade, já que os gradientes não são necessários.

**Loop de Avaliação em Batches:** Para cada lote de validação:

- Serão movidos os dados de entrada e rótulos para o dispositivo de execução (GPU ou CPU).
- Uma passagem direta (*forward pass*) será executada para obter as previsões do modelo.
- As previsões serão calculadas e comparadas com os rótulos para a contagem de acertos.

**Cálculo da Acurácia:** Ao final do loop, a acurácia será calculada e serão divididos o total de acertos pelo número de amostras no conjunto de validação.

**Exibição dos Resultados:** Será exibida a acurácia final no conjunto de validação, ajudando a avaliar o desempenho do modelo.

Após definir a função, ela é chamada para avaliar o modelo treinado.

As instruções acima poderão ser observadas através das linhas de comandos que serão descritas logo abaixo:

```
def evaluate_model(model, dataloaders, device):
    model.eval() # Colocar o modelo em modo de avaliação
    running_corrects = 0

    with torch.no_grad():
        for inputs, labels in dataloaders['val']:
            inputs = inputs.to(device)
            labels = labels.to(device)
```

```

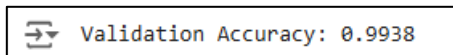
        # Forward pass
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        running_corrects += torch.sum(preds == labels.data)

    accuracy = running_corrects.double() /
len(dataloaders['val'].dataset)
    print(f'Validation Accuracy: {accuracy:.4f}')

# Avaliar o modelo no conjunto de validação
evaluate_model(model, dataloaders, device)

```

E como saída, podemos determinar que para o modelo de avaliação, o valor da acurácia é de 99,38% conforme ilustrado na imagem abaixo:



Validation Accuracy: 0.9938

Figura 9: Valor da acurácia para o modelo de avaliação.

## Avaliando o modelo com os dados de validação

O código a seguir irá definir duas funções para avaliar o modelo com métricas adicionais e para visualizar o desempenho usando uma matriz de confusão e um relatório de classificação detalhando:

**Matriz de Confusão:** A função *plot\_confusion\_matrix* irá calcular a matriz de confusão entre rótulos verdadeiros e predições, visualizando-a por meio de um mapa de calor para mostrar acertos e erros por classe.

**Função de Avaliação com Métricas:** A função *evaluate\_model\_with\_metrics* colocará o modelo em modo de avaliação e desativará o cálculo de gradientes, para que se possa avaliar o modelo de forma mais eficiente.

**Predição e Contagem de Acertos:** Para cada lote de validação, serão armazenadas as predições e os rótulos verdadeiros, e serão contados os acertos.

**Cálculo da Acurácia:** A acurácia total será calculada no conjunto de validação e o resultado exibido.

**Visualização e Relatório:** Será chamado o *plot\_confusion\_matrix* para a visualização da matriz de confusão e será exibido um relatório de classificação que serão resumidas a precisão, revocação e F1-score para cada classe.

Estas instruções poderão ser observadas através das linhas de comandos que serão descritas a seguir:

```
def plot_confusion_matrix(y_true, y_pred, class_names):
```

```

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Confusion Matrix')
plt.show()

def evaluate_model_with_metrics(model, dataloaders, device,
class_names):
    model.eval() # Colocar o modelo em modo de avaliação
    running_corrects = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in dataloaders['val']:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            running_corrects += torch.sum(preds == labels.data)

    accuracy = running_corrects.double() /
len(dataloaders['val'].dataset)
    print(f'Validation Accuracy: {accuracy:.4f}')

    # Calcular e plotar a matriz de confusão
    plot_confusion_matrix(all_labels, all_preds, class_names)

    # Relatório de classificação
    print(classification_report(all_labels, all_preds,
target_names=class_names))

# Chame a função de avaliação com métricas
class_names = dataloaders['train'].dataset.classes # Obtém os
nomes das classes
evaluate_model_with_metrics(model, dataloaders, device,
class_names)

```

E como saída, podemos observar o valor da acurácia com os dados de validação (Figura 10), a matriz de confusão (Figura 11) e o relatório de classificação (Figura 12) que exibe

os valores de precisão (precision), revocação (recall), F1-score e suporte (support), para cada grupo de frutas e para os valores de acurácia, Macro avg e Weighted avg.


 Validation Accuracy: 0.9980

Figura 10: Valor de acurácia com os dados de validação.

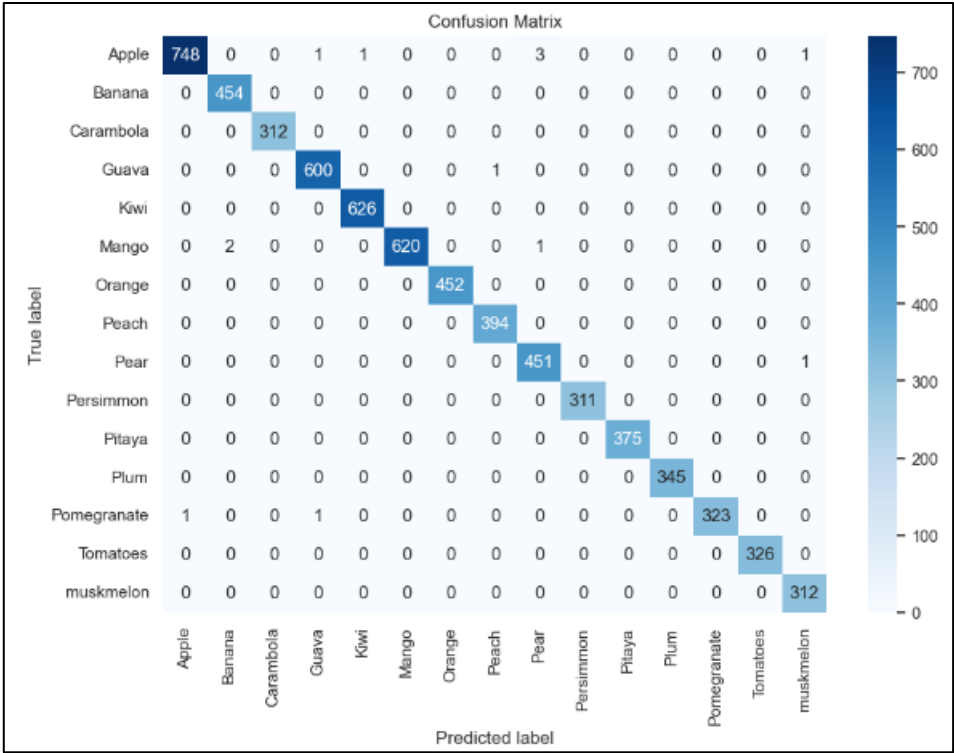


Figura 11: Matriz de Confusão do modelo com os dados de validação.

	precision	recall	f1-score	support
Apple	1.00	0.99	1.00	754
Banana	1.00	1.00	1.00	454
Carambola	1.00	1.00	1.00	312
Guava	1.00	1.00	1.00	601
Kiwi	1.00	1.00	1.00	626
Mango	1.00	1.00	1.00	623
Orange	1.00	1.00	1.00	452
Peach	1.00	1.00	1.00	394
Pear	0.99	1.00	0.99	452
Persimmon	1.00	1.00	1.00	311
Pitaya	1.00	1.00	1.00	375
Plum	1.00	1.00	1.00	345
Pomegranate	1.00	0.99	1.00	325
Tomatoes	1.00	1.00	1.00	326
muskmelon	0.99	1.00	1.00	312
accuracy			1.00	6662
macro avg	1.00	1.00	1.00	6662
weighted avg	1.00	1.00	1.00	6662

Figura 12: Tabela de valores de precisão (precision), revocação (recall), F1-score e suporte (support), para cada grupo de frutas e para os valores de acurácia, Macro avg e Weighted avg.

Conforme a performance do modelo utilizando obtivemos as seguintes análises:

- **De acordo com o relatório de classificação**

- **Acurácia:** 99,80%, que indica a proporção de predições corretas.
- **Precisão e Recall:** Métricas detalhadas para cada classe de fruta, verificando a exatidão e abrangência.
- **F1-Score:** A média harmônica da precisão e revocação, que é alta (1.00) para todas as classes, exceto "Apple" e "Pear", onde é 0.99.
- **Suporte (support):** O número de amostras reais de cada classe. Varia entre 311 e 754 amostras por classe.

Na parte inferior, temos métricas gerais:

- **Acurácia:** 1.00, indicando que o modelo classificou corretamente quase todas as amostras.
- **Macro avg e Weighted avg:** Ambos apresentam valores próximos de 1.00, refletindo o desempenho consistente e equilibrado do modelo em todas as classes.

- **De acordo com a Matriz de Confusão:**

- **Diagonal Principal:** A maioria dos valores está concentrada na diagonal principal, indicando que o modelo classificou corretamente a maioria das amostras para cada classe. Por exemplo, "Apple" foi corretamente classificada 748 vezes, "Banana" 454 vezes, e assim por diante.
- **Erros de Classificação:** Há poucos erros fora da diagonal. Por exemplo:
  - Uma amostra de "Apple" foi classificada como "Mango" e três como "Pear".
  - Uma amostra de "Pomegranate" foi classificada incorretamente como "Apple".
- **Acurácia Geral:** A matriz de confusão confirma a alta acurácia observada no relatório de classificação, com pouquíssimas confusões entre classes.
- **Classes com Pequenos Erros:** "Apple" e "Pomegranate" apresentam algumas confusões, o que está alinhado com o leve impacto na revocação para essas classes no relatório de classificação anterior.

## **Modelo com Base na Função F1-Score**

Utilizaremos um código que irá definir e executar uma função para avaliar o modelo com base no F1-score, que é uma métrica de desempenho que combina precisão e revocação. E para isso utilizaremos as seguintes etapas:

**Modo de Avaliação:** A função `evaluate_model_with_f1` colocará o modelo em modo de avaliação (`model.eval()`), desativando o cálculo de gradientes e camadas como dropout para uma avaliação consistente.

**Armazenamento de Predições e Rótulos:** Inicializará as listas (`all_preds` e `all_labels`) para armazenar todas as predições e rótulos verdadeiros durante a avaliação.

**Loop de Batches:** Para cada lote de dados de validação ocorrerá:

- Transferência das entradas e rótulos para o dispositivo de execução (GPU ou CPU).
- Realização de uma passagem direta (*forward pass*) para obter as previsões.
- Converte as previsões e rótulos para o formato de numpy e armazena-os nas listas.

**Cálculo do F1-Score:** Após coletar todas as predições e rótulos, serão calculados o F1-score ponderado usando `f1_score` do sklearn, o que considerará o peso de cada classe na métrica global.

**Exibição do F1-Score:** Serão exibidos os valores do F1-score final, fornecendo uma métrica única e balanceada de desempenho para o modelo no conjunto de validação.

Essas instruções poderão ser observadas através das linhas de comandos que serão inseridas a seguir:

```
def evaluate_model_with_f1(model, dataloaders, device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in dataloaders['val']:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    f1 = f1_score(all_labels, all_preds, average='weighted')
    print(f'F1 Score: {f1:.4f}')
```



Os comandos a seguir define uma função chamada *evaluate\_model\_with\_precision\_recall*, que é usada para avaliar um modelo de aprendizado de máquina com base nas métricas de precisão (precision) e revocação (recall) no conjunto de dados de validação. Os comandos serão mais bem detalhados a seguir:

### **Definição da Função e Preparação para Avaliação:**

A função *def evaluate\_model\_with\_precision\_recall(model, dataloaders, device)* define uma função que recebe três parâmetros: o modelo (*model*), os carregadores de dados (*dataloaders*), e o dispositivo (*device*, que geralmente será *cpu* ou *cuda* para GPU).

Utilizamos o *model.eval()* com o intuito de colocar o modelo em modo de avaliação, desativando funcionalidades específicas do treinamento (como dropout), para garantir que a avaliação seja estável.

### **Inicialização de Listas para Predições e Labels Verdadeiras:**

Utilizamos o *all\_preds* e o *all\_labels* para que duas listas vazias sejam criadas para armazenar as previsões feitas pelo modelo e os rótulos (labels) reais do conjunto de dados de validação, respectivamente.

### **Laço de Avaliação (Loop):**

Utilizamos o *with torch.no\_grad()* para ativar um contexto sem cálculo de gradientes, reduzindo o consumo de memória e acelerando a execução, pois não precisamos calcular gradientes na fase de avaliação.

No caso, o *for inputs, labels in dataloaders['val']* é empregado para percorrer o conjunto de dados de validação (supostamente identificado pela chave 'val' dentro de *dataloaders*).

Empregamos o *inputs.to(device)* e o *labels.to(device)* para enviar as entradas e rótulos para o dispositivo especificado (CPU ou GPU).

Neste caso, fazemos uso do *outputs = model(inputs)* para fazer a previsão usando o modelo.

E, por fim, o *\_, preds = torch.max(outputs, 1)* tem por objetivo obter a previsão final (a classe com a maior pontuação) de cada amostra.

### **Armazenamento de Predições e Rótulos:**

Os comandos *all\_preds.extend(preds.cpu().numpy())* e *all\_labels.extend(labels.cpu().numpy())* convertem as previsões e rótulos para numpy e armazenam-os nas listas correspondentes.

### Cálculo de Métricas de Precisão e Revocação:

O `precision = precision_score(all_labels, all_preds, average='weighted')` calcula a precisão ponderada (weighted precision), levando em consideração o desequilíbrio de classes.

O `recall = recall_score(all_labels, all_preds, average='weighted')` calcula a revocação ponderada (weighted recall).

E o `print(f'Precision: {precision:.4f}, Recall: {recall:.4f}')` exibe as métricas de precisão e revocação com quatro casas decimais.

Todas estas linhas de comandos poderão ser observadas na programação completa que será inserida a seguir:

```
def evaluate_model_with_precision_recall(model, dataloaders,
device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in dataloaders['val']:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

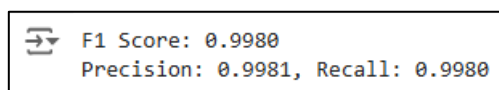
    precision = precision_score(all_labels, all_preds,
average='weighted')
    recall = recall_score(all_labels, all_preds,
average='weighted')
    print(f'Precision: {precision:.4f}, Recall: {recall:.4f}')
```

Como podemos observar, essa função avalia o desempenho do modelo no conjunto de validação, mostrando a precisão e revocação das previsões feitas, o que ajuda a entender a performance geral do modelo em diferentes classes do conjunto de dados.

O código a seguir realizará a avaliação de um modelo no conjunto de validação, utilizando duas funções distintas para calcular métricas de desempenho: uma função para calcular a métrica F1 (*evaluate\_model\_with\_f1*) e outra para calcular precisão e revocação (*evaluate\_model\_with\_precision\_recall*). Conforme será observado na programação a seguir:

```
# Avaliar o modelo no conjunto de validação com métricas
evaluate_model_with_f1(model, dataloaders, device)
evaluate_model_with_precision_recall(model, dataloaders, device)
```

Como saída, teremos a seguinte informação, ilustrada na imagem abaixo:



```
➡ F1 Score: 0.9980
Precision: 0.9981, Recall: 0.9980
```

Figura 13: Valores de saída com os resultados de avaliação do modelo obtido através de F1 Score, Precisão (Precision) e Revocação (Recall).

Essas métricas indicam que:

**F1 Score:** 0.9980 – a média harmônica entre precisão e revocação, indicando um excelente equilíbrio entre ambas, muito próximo de 1 (ou seja, desempenho quase perfeito).

**Precisão (Precision):** 0.9981 – a proporção de previsões corretas entre todas as previsões positivas feitas pelo modelo.

**Revocação (Recall):** 0.9980 – a proporção de previsões corretas entre todos os exemplos positivos reais.

## 10. REALIZAÇÃO DE TRANSFORMAÇÃO E AVALIAÇÃO DO UTILIZANDO COMO BASE DE TESTE A BASE DE DADOS TRATADAS QUE SERÃO EMPREGADAS NO PROJETO

### Definição do Caminho para o Diretório de Teste:

Neste novo conjunto de teste de imagem, criamos um dataset, em que já tratamos as imagens. Deste modo, foi definido um novo caminho para o diretório de teste, que poderá ser observado na linha de comando abaixo:

```
# Definindo o caminho para o diretório de teste
test_dir =
"C:/Users/samue/OneDrive/Documentos/PROJETO_APLICADO/new_dataset_fr
utas/test"
```

O comando acima especifica o caminho para o diretório que contém as imagens de teste do conjunto de dados de frutas.

### Definição das Transformações para as Imagens:

Define uma série de transformações para serem aplicadas às imagens:

- No Redimensionamento, ajusta-se todas as imagens para o tamanho de 224x224 pixels.
- Na Conversão para Tensor, transforma-se as imagens em tensores, que são o formato adequado para manipulação pelo modelo.
- E, na Normalização, ajusta-se os valores dos pixels com médias e desvios padrão específicos, que são comuns para modelos treinados em imagens do ImageNet.

Todas estas instruções poderão ser observadas nas linhas de comando inseridas abaixo:

```
# Defina as transformações a serem aplicadas às imagens
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Redimensiona as imagens
    transforms.ToTensor(), # Converte para tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]), # Normaliza
])
```

### Criação do DataLoader para o Conjunto de Teste

Usamos o *ImageFolder* para carregar as imagens do diretório de teste, aplicando as transformações definidas.

Criamos um *DataLoader* com *batch\_size=32* para processar as imagens em lotes de 32, sem embaralhar os dados (*shuffle=False*), pois isso não é necessário para a fase de teste.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
# Criar o DataLoader para o conjunto de teste
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=32,
shuffle=False) # Não embaralhe os dados para teste
```

### Avaliação do Modelo

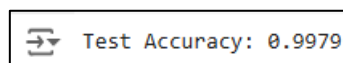
Empregamos a função `evaluate_model` para avaliar o desempenho do modelo no conjunto de teste (`test_loader`), utilizando o dispositivo especificado (`device`), que pode ser CPU ou GPU.

Nesta etapa realizamos a separação dos dados em pastas de treino, validação e teste sendo 69.98% para treino 15% para validação e 15.02% para teste. Dentro das pastas mencionadas existem subpastas separadas em categorias de frutas, elas incluem maçã, banana, manga, goiaba etc.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
# Avaliação do modelo
evaluate_model(model, test_loader, device)
```

E, como saída, a programação do modelo de teste gerou uma acurácia de 99,79%, conforme ilustrada na imagem abaixo:

A screenshot of a terminal window showing the output of the evaluation process. It displays a green icon of a terminal and the text "Test Accuracy: 0.9979".

Test Accuracy: 0.9979

Figura 14: Valor de acurácia do modelo de teste da nova base de dados.

A imagem indica que a precisão do modelo no conjunto de teste é de **99,79%**. Esse valor de precisão (*Test Accuracy: 0.9979*) significa que o modelo acertou aproximadamente 99,79% das classificações ao ser avaliado com o conjunto de dados de teste, o que indica um excelente desempenho do modelo na tarefa para a qual foi treinado.

## 11. CONJUNTO DE TESTE UTILIZANDO MÉTRICAS ADICIONAIS COM GERAÇÃO DE MATRIZ DE CONFUSÃO E RELATÓRIO DE CLASSIFICAÇÃO

Utilizando os dados tratados, e assim como no conjunto de Teste acima, decidimos configurar a avaliação do modelo com métricas adicionais e gerar a matriz de confusão e o relatório de classificação, que serão descritos de forma detalhada logo abaixo:

### Criação do DataLoader para o Conjunto de Teste

Utilizamos o *ImageFolder* para carregar as imagens do diretório de teste e aplicar as transformações configuradas.

Criamos um *DataLoader* para o conjunto de teste, com *batch\_size=32* e sem embaralhar os dados (*shuffle=False*), pois a ordem dos dados não precisa ser aleatória durante o teste.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
# Criar o DataLoader para o conjunto de teste
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=32,
shuffle=False) # Não embaralha os dados para teste
```

### Definição da Função para Plotar a Matriz de Confusão

Através da instrução *plot\_confusion\_matrix* a função que recebe os rótulos verdadeiros (*y\_true*), as previsões (*y\_pred*) e os nomes das classes (*class\_names*).

Calculamos a matriz de confusão usando a instrução *confusion\_matrix* que plota essa matriz através do *seaborn.heatmap*.

A matriz de confusão ajuda a visualizar o desempenho do modelo em cada classe, mostrando onde o modelo acerta e onde ele confunde as classes.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
def plot_confusion_matrix(y_true, y_pred, class_names):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')
    plt.show()
```

### Definição da Função de Avaliação com Métricas Adicionais

A função *evaluate\_model\_with\_metrics* avalia o modelo no conjunto de teste e gera métricas adicionais.

A instrução *running\_corrects* armazena o número total de previsões corretas.

As instruções *all\_preds* e *all\_labels* armazenam, respectivamente, todas as previsões e rótulos verdadeiros para análise posterior.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
def evaluate_model_with_metrics(model, dataloader, device,
                                class_names):
    model.eval() # Colocar o modelo em modo de avaliação
    running_corrects = 0
    all_preds = []
    all_labels = []
```

### Loop de Avaliação

No loop de avaliação, cada lote de dados é processado sem cálculo de gradientes (*torch.no\_grad()*), acelerando a execução.

Para cada lote, o modelo realiza uma previsão, e os rótulos previstos e reais são armazenados em *all\_preds* e *all\_labels*, respectivamente.

Na instrução *running\_corrects* é atualizado com o número de previsões corretas em cada lote.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
with torch.no_grad():
    for inputs, labels in dataloader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
        running_corrects += torch.sum(preds == labels.data)
```

### Cálculo e Impressão da Precisão

A precisão é calculada dividindo o número total de acertos pelo número total de amostras no conjunto de teste, e o valor é exibido.

Essa instrução poderá ser observada na linha de comando inserida logo abaixo:

```
accuracy = running_corrects.double() / len(dataloader.dataset)
print(f'Test Accuracy: {accuracy:.4f}')
```

### Cálculo e Plotagem da Matriz de Confusão

Após o cálculo das previsões, a função *plot\_confusion\_matrix* é chamada para gerar a matriz de confusão entre rótulos verdadeiros e previsões.

Essa instrução poderá ser observada nas linha de comando inserida logo abaixo:

```
# Calcular e plotar a matriz de confusão
plot_confusion_matrix(all_labels, all_preds, class_names)
```

### Geração do Relatório de Classificação

A instrução *classification\_report* exibe métricas detalhadas (precisão, revocação e F1 Score) para cada classe, facilitando a análise do desempenho do modelo em cada uma. Essa instrução poderá ser observada nas linha de comando inserida logo abaixo:

```
# Relatório de classificação
print(classification_report(all_labels, all_preds,
target_names=class_names))
```

### Chamando a Função de Avaliação

A instrução *class\_names* obtém os nomes das classes do conjunto de teste.

A função *evaluate\_model\_with\_metrics* é chamada, utilizando o modelo, o *DataLoader* de teste, o dispositivo (*device*) e os nomes das classes para realizar a avaliação.

Essas instruções poderão ser observadas nas linhas de comandos inseridas logo abaixo:

```
# Chamando a função de avaliação com métricas no conjunto de teste
class_names = test_dataset.classes # Obtém os nomes das classes do
conjunto de teste
evaluate_model_with_metrics(model, test_loader, device,
class_names)
```

E, como saída, teremos o valor de teste de acurácia do modelo (Figura 15), a matriz de confusão (Figura 16) e o relatório de classificação com os valores de precisão (precision), revocação (recall), f1-score e support para os grupos de frutas, acurácia, macro avg e weighted avg (Figura 17). Que serão ilustradas nas imagens a seguir:

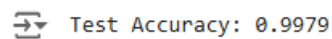
A image shows a small box containing a blue icon of a document with a checkmark, followed by the text "Test Accuracy: 0.9979".

Figura 15: Valor de acurácia do modelo de teste utilizando a base de dados tratada.



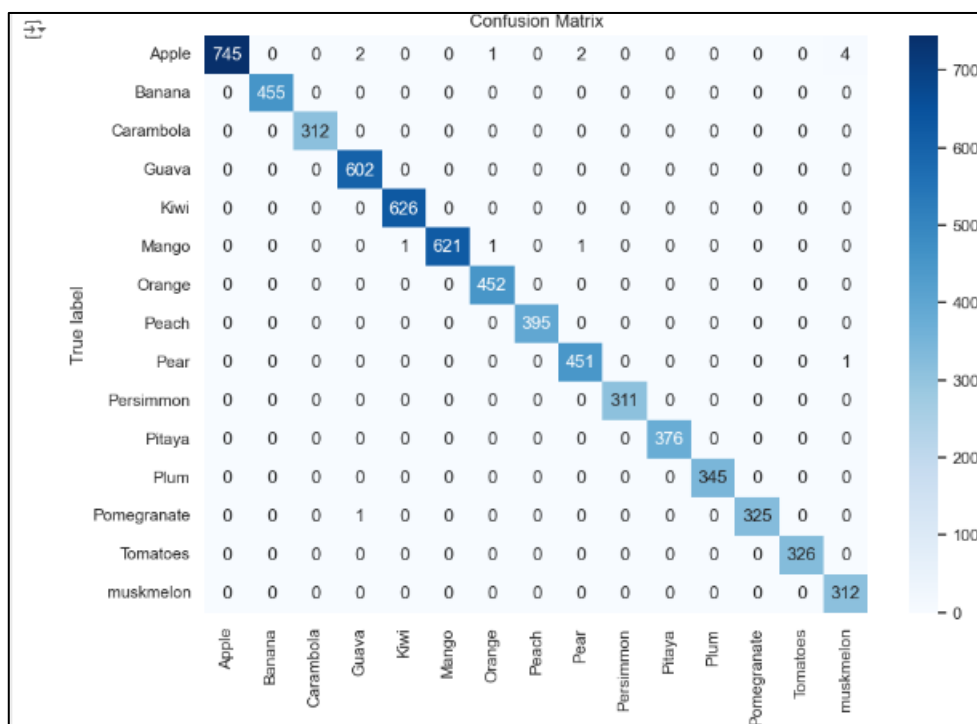


Figura 16: Matriz de Confusão do modelo de Teste utilizando a base de dados tratada.

	Predicted label			
	precision	recall	f1-score	support
Apple	1.00	0.99	0.99	754
Banana	1.00	1.00	1.00	455
Carambola	1.00	1.00	1.00	312
Guava	1.00	1.00	1.00	602
Kiwi	1.00	1.00	1.00	626
Mango	1.00	1.00	1.00	624
Orange	1.00	1.00	1.00	452
Peach	1.00	1.00	1.00	395
Pear	0.99	1.00	1.00	452
Persimmon	1.00	1.00	1.00	311
Pitaya	1.00	1.00	1.00	376
Plum	1.00	1.00	1.00	345
Pomegranate	1.00	1.00	1.00	326
Tomatoes	1.00	1.00	1.00	326
muskmelon	0.98	1.00	0.99	312
accuracy			1.00	6668
macro avg	1.00	1.00	1.00	6668
weighted avg	1.00	1.00	1.00	6668

Figura 17: Relatório de classificação com os valores de precisão (precision), revocação (recall), f1-score e support para os grupos de frutas, acurácia, macro avg e weighted avg, do modelo de Teste utilizando a base de dados tratada.

A imagem da Figura 15 indica que o modelo alcançou uma **precisão de teste de 99,79%** (representada como Test Accuracy: 0.9979). Esse resultado significa que o modelo classificou corretamente aproximadamente 99,79% das amostras para este conjunto de teste.

A imagem da Figura 16 mostra a **matriz de confusão** para o modelo de classificação, avaliando seu desempenho na classificação de diferentes classes de frutas. Cada linha representa a **classe verdadeira** da amostra, e cada coluna representa a **classe prevista** pelo modelo.

Podemos algumas observações sobre os resultados:

**Diagonais Principais:** A maioria dos valores está concentrada na diagonal principal (de cima à esquerda até embaixo à direita), o que indica que o modelo acertou a maioria das classificações. Por exemplo:

- "Apple" tem 745 amostras corretamente classificadas.
- "Banana" tem 455 amostras corretamente classificadas.
- "Guava" tem 602 amostras corretamente classificadas.
- E assim por diante para as outras classes, com números altos nas diagonais, indicando precisão elevada em todas as classes.

**Erros de Classificação:** Valores fora da diagonal representam previsões incorretas. Observando a matriz, vemos que os erros são mínimos, com apenas alguns valores esparsos fora da diagonal principal. Por exemplo:

- Há 2 amostras de "Apple" classificadas incorretamente como "Mango".
- Há 1 amostra de "Guava" classificada incorretamente como "Kiwi".
- Esses erros são pequenos e distribuídos, sugerindo que o modelo confunde poucas amostras entre as classes.

**Desempenho Geral:** A matriz de confusão para este modelo de teste confirma o alto desempenho do modelo, conforme indicado pela precisão de teste de 99,79%. O baixo número de erros sugere que o modelo é muito eficaz na classificação das diferentes frutas e raramente se confunde entre as classes.

A imagem da Figura 17 exibe um **relatório de classificação** com as métricas de avaliação do modelo para cada classe de fruta. As métricas principais incluem **precisão (precision)**, **revocação (recall)**, **F1-score** e **suporte (support)**. Aqui está uma análise detalhada:

**Métricas por Classe:**

- Cada linha representa uma classe de fruta (por exemplo, "Apple", "Banana", etc.), e as colunas mostram as métricas para cada classe.
- **Precisão:** A precisão de todas as classes é próxima de 1.00, indicando que o modelo é muito preciso ao classificar corretamente cada classe.
- **Revocação:** O valor de revocação também é próximo de 1.00 para quase todas as classes, mostrando que o modelo consegue identificar a maioria dos exemplos verdadeiros para cada classe.

- **F1-score:** O F1-score é uma média harmônica da precisão e revocação, e seus valores próximos de 1.00 para todas as classes indicam um excelente equilíbrio entre precisão e revocação.
- **Suporte:** O suporte indica o número total de amostras em cada classe. As classes têm diferentes quantidades de amostras, mas o modelo obteve um desempenho elevado em todas.

#### **Métricas Gerais:**

- **Accuracy:** A precisão geral (accuracy) do modelo é 1.00, indicando que ele acertou todas (ou quase todas) as previsões no conjunto de teste.
- **Macro Avg:** A média macro (macro avg) calcula a média das métricas de precisão, revocação e F1-score sem levar em conta o peso de cada classe. Aqui, a média é de 1.00, indicando que o desempenho do modelo é consistentemente alto em todas as classes.
- **Weighted Avg:** A média ponderada (weighted avg) leva em consideração o suporte de cada classe. Ela também é 1.00, o que confirma que, mesmo levando em conta o desequilíbrio de classes, o modelo tem um excelente desempenho global.

#### **Desempenho em Classes Específicas:**

- A única exceção é a classe "muskmelon", onde a precisão é 0.98, mas ainda com revocação de 1.00, o que indica que o modelo teve apenas uma ligeira dificuldade com esta classe, embora o desempenho ainda seja muito alto.

O relatório mostra que o modelo teve um **desempenho quase perfeito** em todas as classes, com precisão, revocação e F1-score muito próximos de 1.00 para cada fruta. Isso indica que o modelo é extremamente eficaz e confiável na classificação de frutas, independentemente da classe.

## **12. MODELO DE NEGÓCIOS**

**Segmento de Clientes:** Supermercados e empresas de logística que necessitam de sistemas automatizados para classificação rápida e precisa de frutas.

**Proposta de Valor:** Reduzir o tempo e o erro na identificação de frutas, aumentando a eficiência e reduzindo desperdícios em operações comerciais.

**Canais de Distribuição:** Sistema SaaS integrado a ERPs, com suporte para personalização para empresas maiores.

**Fontes de Receita:** Assinaturas mensais, taxas de integração e personalização.

**Parcerias Chave:** Fornecedores de soluções de automação e integradores de ERP no setor de alimentos.

### **13. EXPECTATIVA PARA AS PROXIMAS ENTREGAS**

O processo de treinamento do modelo durou cerca de 8 horas e os resultados apresentados foi bastante satisfatório, tendo uma acurácia máxima de 99,79 para os dados de teste. As métricas utilizadas para a avaliação incluem matriz de confusão, precisão, recall e f1-score.

A expectativa desse projeto tem como meta a criação de um software que diminua o trabalho exaustivo na identificação de frutas, contribuindo na conscientização ambiental, educacional e comercial, disponibilizando informações concretas a partir de parâmetros testados e validados.

### **14. CONCLUSÃO**

O projeto de reconhecimento automático de frutas com redes neurais convolucionais, utilizando a arquitetura ResNet50 com Transfer Learning, apresentou resultados expressivos e mostra grande potencial para aplicação comercial. Com um vasto conjunto de 44.406 imagens, o modelo foi treinado e avaliado com rigor, obtendo uma precisão média de 99,79% no conjunto de testes, resultado que atesta sua confiabilidade e adequação para tarefas de classificação complexas.

A execução do pré-processamento e o detalhado tratamento dos dados, que incluiu a normalização e a separação em conjuntos de treino, validação e teste, garantiram que o modelo fosse capaz de lidar com a diversidade visual presente nas frutas. Além disso, o uso de métricas de avaliação como precisão, recall e F1-score, juntamente com a matriz de confusão, evidenciou um desempenho equilibrado, com excelente capacidade de generalização e baixa ocorrência de erros, mesmo para classes com semelhança visual.

A alta acurácia alcançada demonstra que a aplicação da solução no mercado é uma alternativa viável e promissora, principalmente para supermercados e empresas de logística que precisam de sistemas de classificação rápida e precisa. A proposta de valor do sistema é clara: reduzir o tempo e os erros na identificação de frutas, melhorando a eficiência e minimizando o desperdício. Em etapas futuras, espera-se transformar essa solução em um software prático, integrado a ERPs, que ofereça acesso direto e personalizável para empresas, com possibilidade de adaptação para outros produtos.

Com esse desenvolvimento, o projeto contribuirá para avanços na automação de processos comerciais, promovendo ganhos em precisão e eficiência operacional e auxiliando na sustentabilidade das operações no setor alimentício.

## 15. LINK PARA O GITHUB

<https://github.com/grupos4g4/PROJAPLIC2>

## 16. REFERÊNCIA BIBLIOGRÁFICA

1. Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network." *2017 international conference on engineering and technology (ICET)*. Ieee, 2017.
2. ALVES, Priscila Mello. "Inteligência artificial e redes neurais." *IPEA: Centro de Pesquisa em Ciência* (2020).
3. Der Kiureghian, Armen, and Ove Ditlevsen. "Aleatory or epistemic? Does it matter?." *Structural safety* 31.2 (2009): 105-112.
4. Dino. "IA: Maioria dos Brasileiros Já Utiliza Assistentes Virtuais." *O Globo*, 2 maio 2023, 17h04, <https://oglobo.globo.com/patrocinado/dino/noticia/2023/05/ia-maioria-dos-brasileiros-ja-utiliza-assistentes-virtuais.ghtml>.
5. Garcin, Camille, et al. "PI@ ntNet-300K: a plant image dataset with high label ambiguity and a long-tailed distribution." *NeurIPS 2021-35th Conference on Neural Information Processing Systems*. 2021.
6. Israr Hussain, ., Qianhua He, Zhuliang Chen, & Wei Xie. (2018). Fruit Recognition dataset (V 1.0) [Data set]. Zenodo. <https://doi.org/10.5281/zenodo.1310165>
7. Kanda, Paul Shekonya, Kewen Xia, and Olanrewaju Hazzan Sanusi. "A deep learning-based recognition technique for plant leaf classification." *IEEE Access* 9 (2021): 162590-162613.
8. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90
9. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
10. LeCun, Yann, Koray Kavukcuoglu, and Clément Faret. "Convolutional networks and applications in vision." *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 2010.
11. Vinagreiro, Michel Andre Lima. *Classificação baseada em espaços de camadas convolucionais de redes CNNs densas*. Diss. Universidade de São Paulo, 2022.
12. Willis, Kathy. *State of the world's plants 2017*. Royal Botanic Gardens Kew, 2017.

13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>