Classificação de Frutas Boas e Estragadas Usando Redes Neurais Convolucionais

André H. S. Silva[1], **Gabriel Y. L. Higuchi**[1], **Rafael. P. Czerniej**[1], **Nathan L. S. Oliboni**[1]

[1]Universidade Estadual do Oeste do Paraná -
CEP – 85819-110 – Cascavel – PR – Brazil

andrehss004@gmail.com, gabriel.higuchi@unioeste.br

rafaelpascz@gmail.com, nathan.oliboni@gmail.com

Abstract. *With increasing demands for quality and efficiency in the food supply chain, automated fruit sorting methods are increasingly necessary. This work presents a solution based on convolutional neural networks (CNNs) to identify the condition of fruits, classifying them between "good" and "spoiled". The developed architecture combines convolutional and pooling layers for feature extraction, followed by dense layers, and achieved an accuracy of approximately 95% on the validation set. This article explores the design decisions, optimizations, and final performance of the model, and discusses the implications of its use in practical applications.*

Resumo. *Com o aumento das demandas por qualidade e eficiência na cadeia de abastecimento de alimentos, métodos automatizados de classificação de frutas são cada vez mais necessários. Este trabalho apresenta uma solução baseada em redes neurais convolucionais (CNNs) para identificar a condição de frutas, classificando-as entre "boas" e "estragadas". A arquitetura desenvolvida combina camadas convolucionais e de pooling para a extração de características, seguidas por camadas densas, e alcançou uma acurácia de aproximadamente 95% no conjunto de validação. Este artigo explora as decisões de projeto, otimizações e desempenho final do modelo, além de discutir as implicações de seu uso em aplicações práticas.*

1. Introdução

O consumo de alimentos frescos, em particular frutas, é fundamental para alimentação e saúde humana, mas o manuseio, transporte e estocagem inadequados podem afetar a qualidade do produto, levando a perdas substanciais tanto para consumidores quanto para produtores e distribuidores. Estima-se que uma parte significativa das frutas colhidas em nível mundial nunca chega ao consumidor em função do desperdício causado pelo apodrecimento das frutas durante o transporte e o armazenamento. Considerando este aspecto, a adoção de sistemas automatizados para inspeção e classificação pode ser uma forte alternativa para detectar, ainda em fases iniciais, frutas apresentando sinais de deterioração, possibilitando uma distribuição mais eficiente e uma diminuição do desperdício ao longo da cadeia de fornecimento.

A tarefa de avaliar visualmente a qualidade de frutas envolve a análise de características como textura, coloração e presença de manchas, que são indicadores de frescor

ou deterioração. Essa atividade, quando adotada em grande escala, se torna impossível devido ao alto custo e ao tempo exigido, especialmente quando as empresas trabalham com grandes volumes de produtos. De tal modo, a inteligência artificial com ênfases nas redes neurais convolucionais (CNNs) é particularmente promissora. As CNNs permitem analisar as imagens e efetuar o reconhecimento automático de padrões visuais, o que possibilita o treinamento do modelo para diferenciar frutas boas de estragadas.

Neste trabalho, desenvolvemos um modelo de CNN configurado para distinguir entre frutas boas e estragadas, sendo classificado portanto como um problema de classificação binária, com duas possíveis classes para classificação: fruta boa ou fruta podre.

2. Arquitetura Utilizada

A solução desenvolvida utiliza uma **rede neural convolucional (CNN)** com a **API Sequencial** do TensorFlow. A rede é composta por três camadas convolucionais, cada uma seguida por uma camada de pooling, e termina com camadas densas (fully connected).

```
!pip install tensorflow tensorflow-gpu opencv-python matplotlib
!pip list
!pip install tensorboard
!pip install -q -U keras-tuner

EPOCAS = 10
from sklearn.metrics import confusion_matrix
import seaborn as sns
import cv2
import tensorflow as tf
import os
import imghdr
import numpy as np
from matplotlib import pyplot as plt
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.callbacks import TensorBoard
import keras_tuner as kt
from kerastuner import Hyperband
```

Figure 1. Definição das bibliotecas para arquitetura.

2.1. Entrada dos dados

Para realizar a entrada dos dados para treinamento, teste e validação da **RNA (Rede Neural Artificial)** desenvolvida ao decorrer do projeto, foi-se realizada a separação dos conjuntos de dados em duas classes:

- **Frutas boas**
- **Frutas podres**

Após a definição e classificação das classes e da quantidade de dados que estão disponíveis para serem usados dentro da RNA, é feito então a separação desse conjunto inteiro de dados, onde uma parte fica para treino, outra para teste e finalmente uma para validação.

```
data = tf.keras.utils.image_dataset_from_directory('drive/MyDrive/content/data/') #Tensorflow coleta as imagens do diretório
class_names = data.class_names
print(class_names) #printa as classes encontradas pelo Keras
data_iterator = data.as_numpy_iterator()  #organiza os dados para um iterador no numpy

batch = data_iterator.next() #batch é a coleção das imagens da base em formato matricial

fig, ax = plt.subplots(ncols=4, figsize=(300,300))
for idx, img in enumerate(batch[0][:4]): #codigo para mostrar exemplares dos datasets
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```

```
Found 9469 files belonging to 2 classes.
['Boas', 'Podres']
```

Figure 2. Determinação das classes.

```
data = data.map(lambda x,y: (x/255, y))
data.as_numpy_iterator().next()

train_size = int(len(data)*.7) # treino sera 70% dos dados
val_size = int(len(data)*.2) # validação 20%
test_size = int(len(data)*.1) # teste será 10%

train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

Figure 3. Separação dos dados.

2.2. Camadas Convolucionais

- A primeira camada convolucional utiliza 16 filtros com kernel de tamanho (3x3), seguidos por uma ativação **ReLU**.
- A segunda camada convolucional utiliza 32 filtros e, da mesma forma, é seguida por uma camada de ativação **ReLU**.
- A terceira camada convolucional utiliza 16 filtros e também aplica a ativação **ReLU**.

A **ReLU** (*Rectified Linear Unit*) é uma função de ativação utilizada em redes neurais, na maior parte das vezes em redes neurais convolucionais (CNNs). Ela tem como principal objetivo introduzir **não-linearidade** no modelo, permitindo que a rede aprenda relações mais complexas entre os dados de entrada e saída. De forma clara a **ReLU** simplesmente retorna o valor de entrada xxx se ele for positivo, e retorna **0** se o valor for negativo. Assim, qualquer valor negativo de xxx é levado para zero.

2.3. Pooling

- Camadas de **MaxPooling2D** são aplicadas após cada convolução para reduzir a dimensionalidade espacial das imagens. O pooling seleciona o valor máximo em submatrizes (2x2) para manter características importantes enquanto reduz o tamanho.

Pooling também é uma operação usada em redes neurais convolucionais (**CNNs**) tendo o objetivo de **reduzir a dimensionalidade** dos mapas de características gerados

pelas camadas convolucionais. Simplificando a representação dos dados sem perder informações importantes, diminuindo o número de parâmetros e cálculos necessários para treinar o modelo. Neste método, o pooling seleciona o valor **máximo** de uma submatriz, ou janela, normalmente de tamanho 2x2 em um mapa de características. Ele mantém apenas o valor máximo dentro daquela região.

2.4. Flatten e Dense

- Após a última camada convolucional, os dados são achatados (flatten) para transformá-los em um vetor.
- Uma camada **Dense** com 96 (determinado por busca de hiperparâmetros) neurônios é adicionada com ativação **ReLU** para processamento das características extraídas.

A camada **Flatten** tem a função de **achatar** (ou converter) a saída de uma camada convolucional ou de pooling, que geralmente é tridimensional, em um vetor unidimensional. Permitindo que esses dados possam ser utilizados por camadas densas ou totalmente conectadas.

- **Por que ela é necessária?**
  - Em redes convolucionais, as saídas das camadas de convolução e pooling são representadas como mapas de características tridimensionais, com dimensões de altura×largura×profundidade (por exemplo, 256×256×3 para uma imagem colorida).
  - No entanto, camadas densas requerem **vetores unidimensionais** como entrada. A camada **Flatten** faz essa conversão. Ela pega as características geradas nas camadas convolucionais e as transforma em um vetor longo de valores para que as próximas camadas possam processá-las.

A camada **Dense** (também chamada de camada totalmente conectada ou **fully connected**) é um tipo de camada onde **cada neurônio da camada está conectado a todos os neurônios da camada anterior**. As camadas Dense são utilizadas no final de redes convolucionais para realizar a classificação ou regressão dos dados, transformando as características extraídas das camadas anteriores em previsões.

- **Funcionamento:**
  - Cada neurônio em uma camada Dense recebe uma entrada de todos os neurônios da camada anterior, aplica uma função de ativação (como ReLU, Sigmoid ou Softmax), e produz uma saída.
  - Cada conexão tem um peso associado que é ajustado durante o treinamento para minimizar o erro nas previsões.
- **Aplicações da camada Dense**:
  - **Classificação**: Em problemas de classificação, a camada Dense final pode ter um neurônio com uma ativação **sigmoid** (para classificação binária) ou vários neurônios com ativação **softmax** (para classificação multiclasse).
  - **Regressão**: Em problemas de regressão, a camada Dense final pode não usar uma função de ativação ou usar uma função linear para prever um valor contínuo.
- **Exemplo de uso da camada Dense**:
  - Se a saída da camada Flatten for um vetor de 4096 valores, uma camada Dense com 128 neurônios receberá esses 4096 valores e aplicará suas funções de ativação para produzir 128 novas saídas.

2.5. Camada de Saída

A camada final contém um neurônio, com ativação **sigmoid**, que mapeia o resultado entre 0 e 1. Valores próximos de 0 indicam frutas boas e valores próximos de 1 indicam frutas podres.

3. Parâmetros Utilizados na Solução

3.1. Função de Custo

Foi utilizada a função de perda Binary Crossentropy, que mede a divergência entre a distribuição das probabilidades previstas pelo modelo e as classes reais. A Binary Crossentropy ajusta os pesos da rede para minimizar o erro entre as previsões e os valores reais.

3.2. Otimizador

Optamos pelo otimizador Adam , que combina as vantagens do Gradient Descent com adaptação da taxa de aprendizado ao longo do treinamento. Adam fornece uma abordagem eficiente e eficaz para otimizar os pesos e vieses de uma rede neural.

3.3. Métrica

Utilizamos a acurácia como métrica de avaliação, que calcula a porcentagem de classificações corretas e permite medir o desempenho de forma intuitiva e direta.

3.4. Variáveis de Entrada e Saída

- **Entrada**: Imagens RGB com dimensão ajustada para **256x256x3** pixels. As imagens foram normalizadas dividindo os valores de pixel por 255 para garantir que todas as entradas estivessem entre 0 e 1.
- **Saída**: Um valor contínuo entre 0 e 1, com a interpretação de que valores próximos a 0 indicam ”boa” e próximos a 1 indicam ”podre”.

3.5. Taxa de Aprendizado e Otimizador

- **Taxa de aprendizado**: O otimizador Adam foi utilizado com sua taxa de aprendizado padrão, que é **0.001**.
- **Demais parâmetros**: Foram utilizadas 10 épocas de treinamento, e os dados foram divididos em 70% para treino, 20% para validação e 10% para teste.

3.6. Busca de hiperparâmetros:

Para a avaliação de hiperparâmetros, foi-se utilizado o Hyperband, o avaliador de hiperparâmetros padrão do Keras, pois ele tem um melhor funcionamento em redes neurais do Keras. Foi-se também colocado um critério de parada, caso a busca de hiperparâmetros não trouxesse melhorias em 5 iterações, esta seria finalizada com o objetivo de reduzir o custo computacional desnecessário.

- **Quantidade de neurônios**: A quantidade mínima de neurônios testada foi de 32 e a quantidade máxima foi de 512, com incrementos de 32 em 32,A quantidade ideal de neurônios encontrada foi de 96.
- **Taxa de aprendizado:** As taxas testadas foram de 0.01, 0.001 e 0.0001 A taxa de aprendizado ideal encontrada foi de 0.001

```python
def model_builder(hp):
  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
  hp_units = hp.Int('Units', min_value=32, max_value=512,step = 32)

  keras.Sequential().add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
  keras.Sequential().add(MaxPooling2D())
  keras.Sequential().add(Conv2D(32, (3,3), 1, activation='relu'))
  keras.Sequential().add(MaxPooling2D())
  keras.Sequential().add(Conv2D(16, (3,3), 1, activation='relu'))
  keras.Sequential().add(MaxPooling2D())
  keras.Sequential().add(Flatten())
  keras.Sequential().add(tf.keras.layers.Dense(units=hp_units, activation='relu'))
  keras.Sequential().add(Dense(1, activation='sigmoid'))
  keras.Sequential().compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate)
                ,loss=keras.losses.BinaryCrossentropy(from_logits=False),metrics=['accuracy'])
  return keras.Sequential()
```

Figure 4. Arquitetura do modelo.

```python
#instanciar hypertuner para o processo de busca de hiperparametros
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy', #aumentar a precisão da predição
                     max_epochs=EPOCAS,
                     factor=3, #fator de regressão, valor padrão é 3
                     directory='drive/MyDrive/content',
                     project_name='RedeNeuralTrabalho',
                     overwrite=True)

stop_early = tf.keras.callbacks.EarlyStoprping(monitor='val_loss', patience=5)
#definir um critério de parada do hypertuner, se em 5 iterações o val_loss não melhorou, ele para

#iniciar busca, mesmos parametros que model.fit
tuner.search(train, epochs=EPOCAS, validation_data=val, callbacks=[stop_early])

best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"""Busca completa, o numero de units para a primeira camada densa é {best_hps.get('Units')}
      e a melhor taxa de aprendizado é {best_hps.get('learning_rate')}.""")
```

Figure 5. Estrutura da Hyperband.

4. Treinamento do modelo

Para o treinamento do modelo, utilizou-se o conjunto de dados previamente dividido e a arquitetura configurada conforme os parâmetros ideais encontrados na busca de hiperparâmetros. O processo de treinamento foi monitorado com a ajuda de callbacks, como o `TensorBoard`, que possibilita a análise visual de métricas de desempenho e ajustes em tempo real, permitindo identificar se o modelo converge para uma boa solução ao longo das épocas.

Durante o treinamento, foram registradas a perda e a acurácia para o conjunto de treino e validação. Isso permitiu verificar o desempenho do modelo e, ao final, visualizar as métricas em gráficos que mostram a evolução de cada época. As curvas de perda (*loss*) e de acurácia (*accuracy*) foram plotadas, evidenciando a eficácia do modelo em aprender a partir dos dados e como ele generaliza ao validar os exemplos de frutas boas e podres.

A métrica de acurácia foi utilizada para avaliar o percentual de classificações corretas em cada época, sendo possível verificar se o modelo se ajustou bem aos dados ou

apresentou sinais de sobreajuste ou subajuste.

5. Resultados

A rede neural conseguiu realizar a tarefa de classificação de frutas boas e podres com os seguintes resultados:

5.1. Percentual de Acertos e Erros

Após o treinamento, a rede neural obteve boas taxas de acurácia tanto no conjunto de treino quanto no de validação. A acurácia final durante o treinamento foi de aproximadamente **95%** no conjunto de treino e **90%** no conjunto de validação, conforme os gráficos gerados a partir do histórico de treinamento.

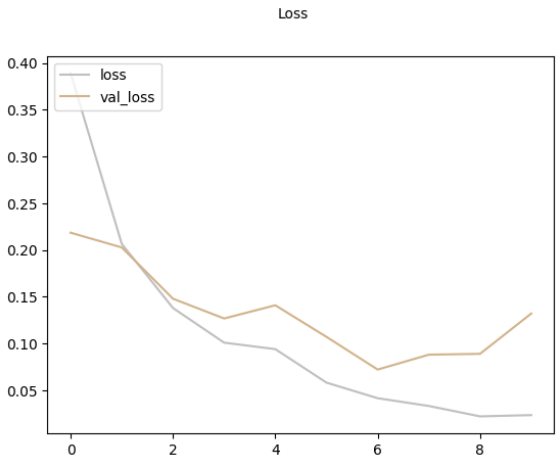

Figure 6. Gráfico de perda.

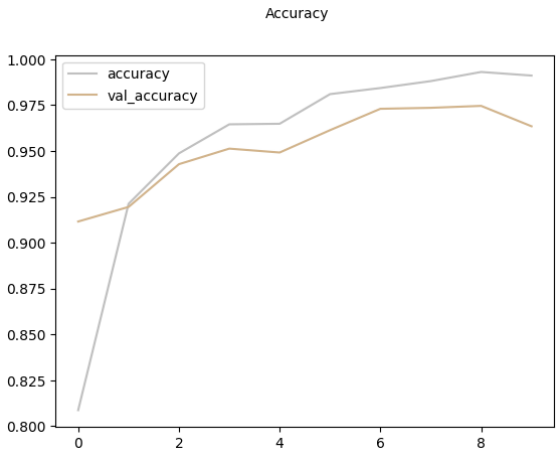

Figure 7. Gráfico de acurácia.

5.2. Tentativas de Ajuste da Rede

- Foram feitos ajustes no número de filtros das camadas convolucionais e na quantidade de neurônios da camada densa para maximizar o desempenho sem sobreajustar (overfitting).

- O uso de camadas de pooling ajudou a reduzir a complexidade do modelo e melhorou o tempo de treinamento, preservando as características mais importantes das imagens.

5.3. Teste com imagens inseridas manualmente

Após a finalização do modelo, treinamento e validação, foi-se realizado então um teste manual, onde foram separadas 150 imagens de frutas, 75 eram boas e 75 eram podres. Determinado então quantas seriam as imagens de cada classe e sua quantidade de imagens, foi-se carregado o modelo, divididas as imagens e carregadas as imagens para dentro do modelo, os resultados obtidos foram:

- **Frutas boas:**

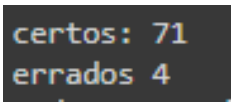

Figure 8. Boas.

- **Frutas podres:**

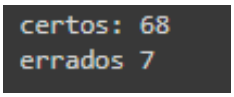

Figure 9. Podres.

5.4. Matriz de Confusão

A matriz de confusão demonstrou que o modelo possui uma taxa baixa de falsos positivos e falsos negativos, o que significa que ele identifica bem tanto as frutas boas quanto as podres.

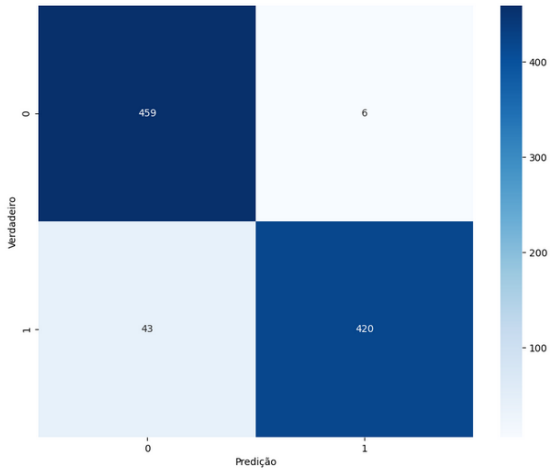

Figure 10. Matriz de confusão.

6. Conclusão

O uso de redes neurais convolucionais (CNNs) para a classificação de frutas entre boas e podres mostrou-se uma abordagem eficiente, alcançando uma acurácia de até 95 por cento no conjunto de treino e 90 por cento na validação, mostrando que o modelo desenvolvido possui um potencial significativo para aplicação em ambientes reais de processamento e triagem de frutas, podendo contribuir para a redução de perdas e desperdícios ao longo da cadeia de abastecimento.

Os resultados destacam que a arquitetura empregada, com camadas convolucionais seguidas por pooling, e uma camada final densa, foi eficaz na extração de características relevantes e na classificação. Através de uma otimização cuidadosa de hiperparâmetros e do uso de callbacks para monitoramento do treinamento, foi possível minimizar o risco de **overfitting**, mantendo uma boa capacidade de generalização.

Direções futuras para este trabalho incluem estender o modelo para outras categorias de deterioração, como machucados ou manchas específicas, e uma análise do seu desempenho em dados sendo coletados em condições reais de campo, onde há maiores variações nas características das imagens. A aplicação de técnicas de aumento de dados também pode ser uma alternativa para melhorar a robustez do modelo. Com a evolução contínua na área de inteligência artificial e o aumento do poder computacional, as CNNs devem se postular como uma ferramenta importante em aplicações práticas de classificação visual, tanto na agricultura como na indústria alimentar, contribuindo para aumentar a eficiência e a sustentabilidade no manejo e distribuição de produtos frescos.

7. Referências

References

ALVES, Gisely. Entendendo Redes Convolucionais (CNNs). 2018. Disponível em: https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184. Acesso em: 14 out. 2024.

BROWLEE, Jason. How to Avoid Overfitting in Deep Learning Neural Networks. 2019. Disponível em: https://www.machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/. Acesso em: 14 out. 2024.

CECCON, Denny. Funções de ativação: definição, características, e quando usar cada uma. 2020. Disponível em: https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/. Acesso em: 14 out. 2024.

IBM. Convolutional Neural Networks. Disponível em: https://www.ibm.com/br-pt/topics/convolutional-neural-networks. Acesso em: 14 out. 2024.

M., Vijay. What is the difference between Training Loss Validation Loss and Evaluation Loss. 2023. Disponível em: https://medium.com/@penpencil.blr/what-is-the-difference-between-training-loss-validation-loss-and-evaluation-loss-c169ddeccd59. Acesso em: 14 out. 2024.

RIBEIRO, Leonardo F.R. O que são Redes Neurais Convolucionais. 2020. Disponível em: https://www.youtube.com/watch?v=sgt$_6$7$zNDrI.Acessoem$: $14out.2024.$

TENSORFLOW (org.). TensorFlow Core: keras. 2020. Disponível em: https://www.tensorflow.org/guide/keras?hl=pt-br. Acesso em: 14 out. 2024.

VICERI (org.). Arquiteturas de Redes Neurais Convolucionais para reconhecimento de imagens. 2020. Disponível em: https://viceri.com.br/insights/arquiteturas-de-redes-neurais-convolucionais-para-reconhecimento-de-imagens/. Acesso em: 14 out. 2024.