Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Engenharia de Computação
Disciplina: IF69D – Processamento Digital de Imagens
Semestre: 2025.1
Prof.: Gustavo Borba e Humberto Gamba

RELATÓRIO
Modelo e instruções (classificação de alimentos com CNN)

Aluno:
Gustavo Adame Domingues / 2280515

junho.2025

1. Objetivo

Este trabalho objetiva o desenvolvimento e a otimização de uma Rede Neural Convolucional (CNN) para a classificação de 16 categorias de alimentos. A metodologia empregada compara um modelo base, treinado com um otimizador fundamental, com uma versão otimizada que utiliza callbacks e técnicas avançadas para demonstrar a melhoria na acurácia e a mitigação do overfitting.

2. Fundamentação Teórica

2.1. Redes Neurais Convolucionais (CNNs)

As Redes Neurais Convolucionais (CNNs) são uma classe de modelos de deep learning que se tornaram a abordagem padrão para tarefas envolvendo dados com estrutura de grade, como imagens digitais. Sua arquitetura é projetada para reconhecer padrões visuais de forma hierárquica, onde camadas iniciais aprendem características simples (e.g., bordas, cores) e camadas subsequentes combinam essas características para identificar formas e objetos complexos.

Uma arquitetura de CNN para classificação, como a implementada neste trabalho, é composta fundamentalmente por um extrator de características seguido por um classificador. A Figura 1 ilustra este fluxo de processamento.

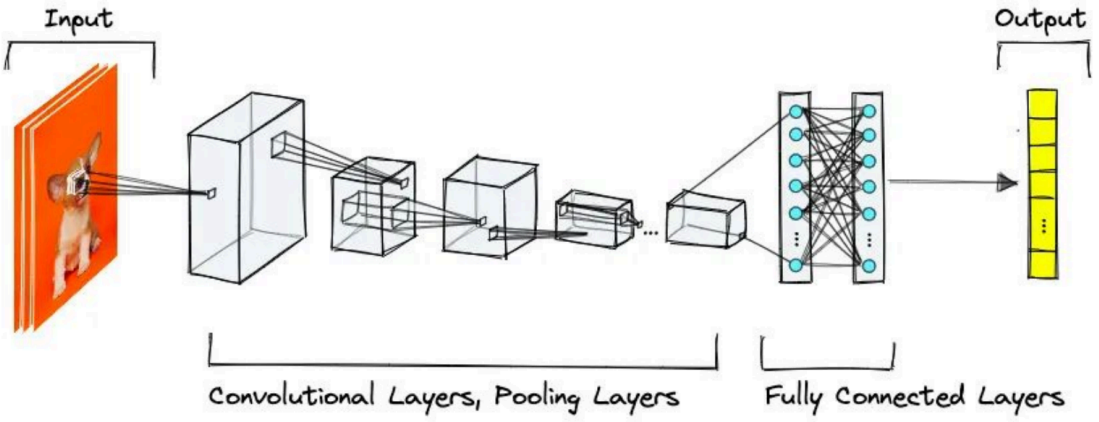

Figura 1 – Fluxo geral de uma rede neural convolucional (CNN) [4]

Os componentes principais dessa arquitetura são as seguintes camadas:

Camada Convolucional: Este é o bloco de construção central de uma CNN. Ele utiliza um conjunto de filtros (ou kernels) que deslizam sobre a imagem de entrada para realizar a operação de convolução. Cada filtro é treinado para se ativar ao detectar um padrão local específico, como uma borda vertical, uma textura ou uma cor. O resultado é um conjunto de mapas de ativação que indicam a presença dessas características na imagem. [3]

Camada de Agrupamento: Geralmente utilizada após uma camada convolucional, a camada de agrupamento serve para reduzir as dimensões espaciais (altura e largura) dos mapas de características. O método mais comum, Max Pooling, seleciona o valor máximo de uma pequena janela, o que diminui a carga computacional e ajuda a criar uma representação que é mais robusta a pequenas variações na posição do objeto. [3]

Camada de Achatamento (Flatten): Atuando como uma ponte entre o extrator de características e o classificador, a função desta camada é transformar o mapa de características 2D final em um único e longo vetor 1D. Esta reorganização dos dados é um passo necessário para que possam ser processados pelas camadas densas subsequentes. [3]O treinamento de uma Rede Neural Convolucional é um processo iterativo cujo objetivo é ajustar os parâmetros internos do modelo (pesos e bias) para minimizar uma função de erro pré-definida. Este ciclo de otimização é orquestrado pela interação entre uma função de perda, que quantifica o erro do modelo, e um otimizador, que atualiza os parâmetros para reduzir esse erro.

Camada Densa (Dense): Após o achatamento, o vetor de características alimenta um classificador, que é tipicamente um Perceptron de Múltiplas Camadas (MLP) composto por uma ou mais camadas densas. Estas são camadas neurais totalmente conectadas, onde cada neurônio está conectado a todos os neurônios da camada anterior. Elas são responsáveis por aprender as combinações complexas das características extraídas para, por fim, classificar a imagem. [3]

Camada Softmax: A última camada densa do modelo utiliza a função de ativação softmax para gerar um vetor de probabilidades, indicando a chance de a imagem de entrada pertencer a cada uma das classes definidas. [3]

2.2. Processo de Treinamento

O treinamento de uma Rede Neural Convolucional é um processo iterativo cujo objetivo é ajustar os parâmetros internos do modelo (pesos e *bias*) para minimizar uma função de erro pré-definida. Este ciclo de otimização é orquestrado pela interação entre uma função de perda, que quantifica o erro do modelo, e um otimizador, que atualiza os parâmetros para reduzir esse erro.

Função de Perda (Loss Function): Para mensurar o quão incorreta é a previsão de um modelo em tarefas de classificação multiclasse, a função de perda Entropia Cruzada Categórica (Categorical Cross-Entropy) é a métrica padrão. Ela atua medindo a divergência entre a distribuição de probabilidade gerada pela camada de saída softmax e a distribuição de probabilidade real da etiqueta (onde a classe correta tem probabilidade 1 e as demais têm 0). Um valor de perda mais alto indica uma maior discrepância entre a previsão e a realidade, sinalizando a necessidade de uma correção mais significativa nos pesos do modelo. [4]

Otimizadores: O mecanismo responsável por minimizar a função de perda é o otimizador. Ele implementa uma variação do algoritmo de Descida do Gradiente (Gradient Descent), que ajusta iterativamente os parâmetros da rede na direção oposta ao gradiente da função de perda. A

computação eficiente desses gradientes em redes profundas é realizada pelo algoritmo de Backpropagation. Neste trabalho, foram utilizados e comparados dois otimizadores distintos:

- **SGD (Stochastic Gradient Descent):** É a implementação clássica da Descida do Gradiente. Sua principal característica é a utilização de uma única taxa de aprendizado (learning rate) fixa para todos os parâmetros do modelo. Embora robusto, sua convergência pode ser mais lenta e sensível à escolha inicial desta taxa. [2]
- **Adam (Adaptive Moment Estimation):** Um otimizador mais avançado e amplamente adotado que se destaca por sua capacidade de computar taxas de aprendizado adaptativas para cada parâmetro individualmente. Ele combina as vantagens de outros otimizadores, como o RMSprop e o SGD com momentum, resultando frequentemente em uma convergência mais rápida e um desempenho mais estável em uma vasta gama de problemas. [2]

2.3. Técnicas de Otimização e Regularização

Além da arquitetura da rede e da escolha do otimizador, foram empregadas técnicas adicionais para melhorar a capacidade de generalização do modelo e combater o overfitting, que é a tendência de um modelo se ajustar excessivamente aos dados de treino, perdendo a capacidade de performar bem em dados novos.

Aumento de Dados (Data Augmentation): Para aumentar artificialmente a diversidade do conjunto de treinamento sem a necessidade de coletar novas imagens, a técnica de aumento de dados foi aplicada. Durante o treinamento, transformações aleatórias como rotações, zoom, cisalhamento e inversões horizontais são aplicadas em tempo real às imagens de treino. Este processo expõe o modelo a uma variedade muito maior de exemplos, forçando-o a aprender as características essenciais dos objetos em vez de memorizar as imagens específicas, o que o torna mais robusto a variações de ângulo e posição.

Camada de Dropout: Como principal técnica de regularização na arquitetura, foi utilizada a camada de Dropout. Durante cada etapa do treinamento, esta camada "desliga" aleatoriamente uma fração pré-definida de neurônios da camada anterior. Ao fazer isso, ela impede que a rede se torne excessivamente dependente de um pequeno conjunto de neurônios para fazer suas previsões. Essa abordagem força a rede a aprender representações mais distribuídas e robustas, melhorando significativamente sua capacidade de generalização para dados não vistos.

Callbacks (Monitoramento e Controle do Treinamento): Os Callbacks são ferramentas que permitem monitorar e intervir no processo de treinamento de forma automatizada. No escopo deste trabalho, estas técnicas avançadas foram aplicadas exclusivamente ao 'Modelo Otimizado' para avaliar seu impacto em comparação com o treinamento padrão de épocas fixas do 'Modelo Base'. Os callbacks utilizados foram:

- **EarlyStopping:** Interrompe o treinamento se a métrica monitorada (neste caso, a perda na validação) não apresentar melhora por um número definido de épocas (patience), evitando o overfitting e economizando tempo computacional.
- **ModelCheckpoint:** Salva em disco a versão do modelo que atingiu o melhor desempenho na métrica monitorada durante todo o processo de treinamento, garantindo que o artefato final seja o modelo mais performático.otimizador Adam
- **ReduceLROnPlateau:** Ajusta dinamicamente a taxa de aprendizado. Se o modelo estagna e não melhora a métrica monitorada, este callback reduz a taxa de aprendizado, permitindo "ajustes finos" para que o otimizador possa encontrar mínimos de perda mais sutis.

3. Implementação

A implementação deste projeto foi desenvolvida na linguagem *Python*, utilizando o *framework TensorFlow* com a sua *API* de alto nível, *Keras*, para a construção e treinamento dos modelos de Rede Neural Convolucional. O desenvolvimento e a experimentação foram conduzidos em um ambiente de *notebook Jupyter*. Esta seção detalha as etapas práticas do projeto, desde a preparação do conjunto de dados até a configuração e treinamento dos modelos propostos. Os sub-tópicos a seguir descrevem a estruturação do *dataset*, a definição das arquiteturas e as configurações específicas de treinamento para os modelos base e otimizado. Instruções detalhadas para a execução do código, bem como a descrição de cada arquivo que compõe o projeto, podem ser encontradas no arquivo *README.md* do repositório público do projeto no *GitHub*: https://github.com/GustavoAdamee/cnn_food.

3.1. Dataset e Pré-processamento

O conjunto de dados utilizado neste trabalho é composto por um total de 1600 imagens, divididas em 16 classes de alimentos, incluindo arroz, batata, tomate, entre outros.

Para treinar e avaliar os modelos de forma robusta, o dataset foi primeiramente dividido em dois subconjuntos distintos: treinamento, contendo 80% das imagens de cada classe, e validação, com os 20% restantes. Esta separação foi realizada de forma automatizada e aleatória por meio de um script customizado em *Python*, o qual é melhor explicado no arquivo *README.md.*

O carregamento das imagens e a aplicação de técnicas de pré-processamento foram gerenciados pela classe ImageDataGenerator da API Keras. Esta ferramenta é eficiente em termos de memória, pois carrega os dados em lotes (batches) diretamente dos diretórios, e permite a aplicação de transformações de aumento de dados (data augmentation) em tempo real, exclusivamente no conjunto de treinamento. A configuração dos geradores é apresentada a seguir na Figura(2).

```python
# Define as dimensões para as quais todas as imagens serão redimensionadas
IMG_HEIGHT = 150
IMG_WIDTH = 150

# Tamanho de cada batch
BATCH_SIZE = 32

TRAIN_DIR = '../processed_data/dataset_1_split/train'
VALIDATION_DIR = '../processed_data/dataset_1_split/validation'


print("Configurando o gerador de dados de TREINO...")
# Gerador de dados para o conjunto de TREINO (com Data Augmentation)
train_datagen = ImageDataGenerator(
    rescale=1./255,             # Normaliza os pixels para o intervalo [0, 1]
    rotation_range=40,          # Rotaciona a imagem em até 40 graus
    width_shift_range=0.2,      # Desloca a largura em até 20%
    height_shift_range=0.2,     # Desloca a altura em até 20%
    shear_range=0.2,            # Aplica cisalhamento
    zoom_range=0.2,             # Aplica zoom de até 20%
    horizontal_flip=True,       # Inverte a imagem horizontalmente
    fill_mode='nearest'         # Preenche pixels novos com o valor mais próximo
)


print("Configurando o gerador de dados de VALIDAÇÃO...")
# Gerador de dados para o conjunto de VALIDAÇÃO (APENAS normalização)
validation_datagen = ImageDataGenerator(
    rescale=1./255
)
```

Figura 2 – Configuração dos geradores

Conforme mostrado no Trecho de Código 1, todas as imagens foram padronizadas para uma dimensão de 150x150 pixels e seus valores de pixel foram normalizados para o intervalo [0, 1] através do parâmetro rescale=1./255, uma etapa essencial para o bom desempenho do treinamento. Adicionalmente, o gerador de treinamento (train_datagen) foi configurado para aplicar as transformações aleatórias mencionadas, aumentando a variabilidade dos dados e auxiliando na regularização do modelo para combater o overfitting.

3.2. Arquiteturas e Configurações

Para avaliar o impacto das otimizações, foram implementados dois modelos distintos: um "**Modelo Base**", com uma arquitetura e configuração fundamentais, e um "**Modelo Otimizado**", que emprega tanto uma arquitetura mais profunda quanto métodos de treinamento mais avançados. Esta abordagem permite analisar os ganhos de performance obtidos tanto pelo aprimoramento da arquitetura quanto pela inteligência do processo de treinamento.

- **Arquitetura do Modelo Base:** A arquitetura da CNN Base consiste em três blocos convolucionais seguidos por um classificador MLP. Cada bloco é composto por uma camada Conv2D com ativação ReLU e padding='same', seguida por uma camada MaxPooling2D. A estrutura detalhada é apresentada na Figura 3.

Layer (type)	Output Shape	Param #
conv1_1 (Conv2D)	(None, 150, 150, 32)	896
pool1 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2_1 (Conv2D)	(None, 75, 75, 64)	18,496
pool2 (MaxPooling2D)	(None, 37, 37, 64)	0
conv3_1 (Conv2D)	(None, 37, 37, 128)	73,856
pool3 (MaxPooling2D)	(None, 18, 18, 128)	0
flatten (Flatten)	(None, 41472)	0
dense1 (Dense)	(None, 256)	10,617,088
dropout (Dropout)	(None, 256)	0
output (Dense)	(None, 16)	4,112

Figura 3 – Resumo da arquitetura do modelo base

- **Arquiterura do Modulo Otimizado:** A arquitetura do Modelo Otimizado utiliza a mesma estrutura do modelo base, mas com uma modificação para aumentar sua capacidade de aprendizado: foi adicionada uma segunda camada Conv2D com 128 filtros ao terceiro bloco convolucional. O objetivo desta alteração foi aprofundar a rede, permitindo que ela aprendesse características de nível mais alto e mais abstratas antes da etapa de classificação. A figura 4 ilustra a modificação neste bloco específico.

```
# --- ARQUITETURA DA CNN MELHORADA---
model = models.Sequential(name='CNN_Alimentos_Parte1')

# --- BLOCO 1 ---
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3), padding='same', name='conv1_1'))
model.add(layers.MaxPooling2D((2, 2), name='pool1'))

# --- BLOCO 2 ---
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same' ,name='conv2_1'))
model.add(layers.MaxPooling2D((2, 2), name='pool2'))

# --- BLOCO 3 ---
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same', name='conv3_1'))
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same', name='conv3_2'))    #Camada extra
model.add(layers.MaxPooling2D((2, 2), name='pool3'))

# --- CLASSIFICADOR (MLP) ---
model.add(layers.Flatten(name='flatten'))

model.add(layers.Dense(256, activation='relu', name='dense1'))

model.add(layers.Dropout(0.5, name='dropout'))

model.add(layers.Dense(16, activation='softmax', name='output'))
```

Figura 4 – Código da implementação da arquitetura base com uma camada extra convolucional no terceiro bloco,

A principal diferença entre so modelos, além da arquitetura, reside na configuração da compilação do modelo.

- **Configuração do Modelo Base:** O Modelo Base foi compilado utilizando o otimizador clássico SGD (Stochastic Gradient Descent) e treinado por um número fixo de 20 épocas, representando uma abordagem mais simples e fundamental.
- **Configuração do Modelo Otimizado:** O Modelo Otimizado foi configurado com o *otimizador Adam* [2], com uma taxa de aprendizado customizada. Além disso, seu treinamento foi gerenciado por Callbacks para parar o processo no momento ideal, salvar apenas a melhor versão e ajustar a taxa de aprendizado dinamicamente.

4. Resultados e conclusões

Nesta seção, são apresentados e analisados os resultados obtidos pelos modelos base e otimizado, comparando seu desempenho de forma quantitativa e qualitativa para avaliar a eficácia das otimizações implementadas.

A Figura 5 apresenta as curvas de aprendizado para os modelos base e otimizado. Para o Modelo Base (Figura 5a), a análise revela um claro sinal de underfitting (subajuste). Este fenômeno é evidenciado no gráfico de perda, onde a perda no conjunto de treino (linha azul) permanece consistentemente superior à perda de validação (linha laranja). Este comportamento indica que o modelo possui dificuldade em aprender até mesmo os padrões dos dados de treinamento, possivelmente devido a uma combinação de uma arquitetura com capacidade limitada para 16 classes e uma regularização forte. A acurácia de validação, que se mostra superior à de treino, é um efeito colateral esperado da camada de Dropout, que está ativa apenas durante a fase de treinamento e não na validação.

Em contraste, o Modelo Otimizado (Figura 5b), beneficiado por um otimizador mais avançado (Adam), uma arquitetura mais profunda e um processo de treinamento inteligente com callbacks, supera o underfitting. Suas curvas de perda são significativamente mais baixas e as curvas de acurácia atingem um patamar muito superior, demonstrando um aprendizado eficaz e uma boa capacidade de generalização.

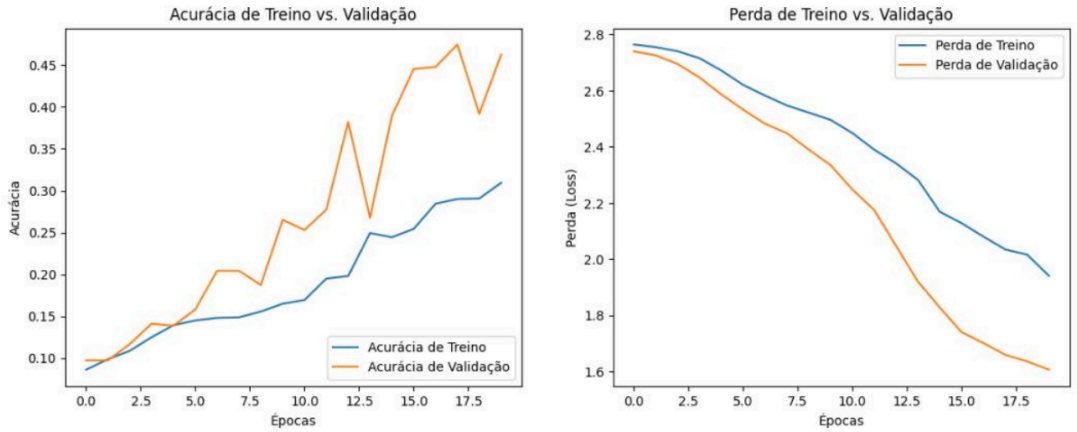

Figura 5a – Curvas de aprendizado do modelo base

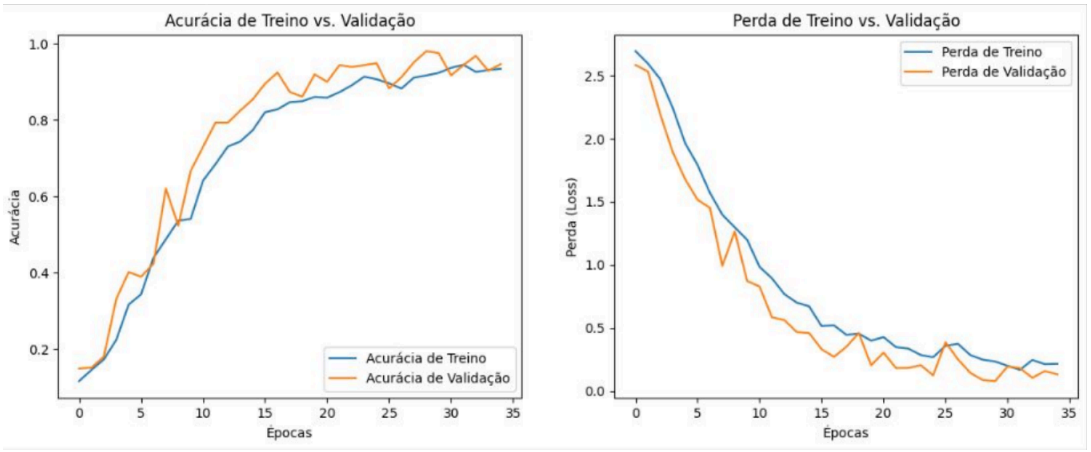

Figura 5b – Curvas de aprendizado do modelo otimizado

A superioridade do modelo otimizado é confirmada numericamente pelos resultados finais no conjunto de validação, sumarizados na Figura 6. O modelo otimizado alcançou uma acurácia final 48 pontos percentuais maior que o modelo base.

	precision	recall	f1-score	support
Alface	1.00	0.33	0.50	24
Almondega	0.00	0.00	0.00	23
Arroz	0.19	0.58	0.29	24
BatataFrita	0.62	0.54	0.58	24
Beterraba	0.49	0.95	0.65	22
BifeBovinoChapa	0.76	0.50	0.60	26
CarneBovinaPanela	0.48	0.78	0.59	40
Cenoura	1.00	0.23	0.37	31
FeijaoCarioca	0.80	0.16	0.27	25
Macarrao	0.52	0.96	0.68	27
Maionese	0.43	0.52	0.47	23
PeitoFrango	0.86	0.75	0.80	24
PureBatata	0.56	0.64	0.60	28
StrogonoffCarne	0.00	0.00	0.00	22
StrogonoffFrango	1.00	0.57	0.73	21
Tomate	0.24	0.26	0.25	27
accuracy			0.50	411
macro avg	0.56	0.49	0.46	411
weighted avg	0.56	0.50	0.47	411

Figura 6a – relatório modela base

	precision	recall	f1-score	support
Alface	1.00	1.00	1.00	24
Almondega	0.96	0.96	0.96	23
Arroz	0.83	1.00	0.91	24
BatataFrita	1.00	1.00	1.00	24
Beterraba	1.00	1.00	1.00	22
BifeBovinoChapa	1.00	1.00	1.00	26
CarneBovinaPanela	1.00	1.00	1.00	40
Cenoura	1.00	1.00	1.00	31
FeijaoCarioca	1.00	1.00	1.00	25
Macarrao	1.00	1.00	1.00	27
Maionese	0.95	0.87	0.91	23
PeitoFrango	0.92	1.00	0.96	24
PureBatata	1.00	0.96	0.98	28
StrogonoffCarne	1.00	0.91	0.95	22
StrogonoffFrango	1.00	0.90	0.95	21
Tomate	1.00	1.00	1.00	27
accuracy			0.98	411
macro avg	0.98	0.98	0.98	411
weighted avg	0.98	0.98	0.98	411

Figura 6b – relatório modelo otimizado

Para uma análise mais profunda do desempenho por classe, a matriz de confusão do Modelo Otimizado é apresentada na Figura 7. A diagonal principal, que representa os acertos, é fortemente populada, indicando uma boa performance geral. Contudo, o modelo demonstrou alguma dificuldade em distinguir entre "Maionese" e "Arroz", que foram ocasionalmente confundidas. Isso pode ser atribuído à alta similaridade visual entre estas classes.

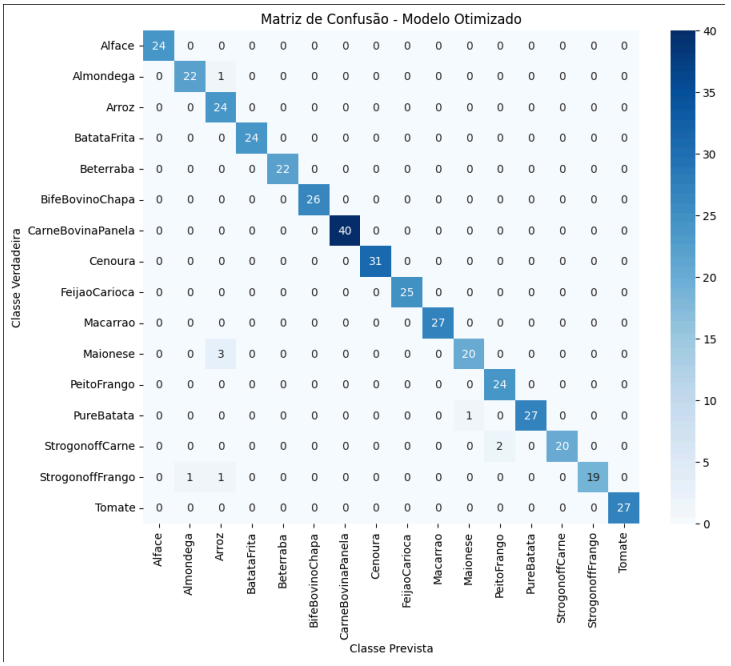

Figura 7 – Matriz de confusão do modelo otimizado

Em análise crítica, os resultados demonstram que a abordagem de otimização foi altamente eficaz. O ganho de performance foi significativo e as técnicas aplicadas, em especial os callbacks de treinamento e o otimizador Adam, foram cruciais para mitigar o underfitting observado no modelo base. As confusões remanescentes indicam que o desafio de distinguir classes com baixa variação inter-classe é um limite do modelo atual, que poderia ser superado com um extrator de características mais potente ou com um dataset ainda mais diverso.

Referências

[1] TensorFlow. “Rede Neural Convolucional (CNN).” **Documentação do TensorFlow**. Disponível em: http://tensorflow.org/tutorials/images/cnn?hl=pt-br. Acesso em: 24 Junho, 2025.

[2] Awan, A. A. “Otimizador Adam: Guia completo para iniciantes.” **DataCamp, 2023**. Disponível em: https://www.datacamp.com/pt/tutorial/adam-optimizer-tutorial. Acesso em: 24 Junho, 2025.

[3] Navlani, A., Apurva, N. “Introduction to Convolutional Neural Networks (CNNs).” **DataCamp, 2023.** Disponível em: https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns. Acesso em: 24 Junho, 2025.

[4] Briggs, J. “Convolutional Neural Networks (CNNs).” **Pinecone**. Disponível em: https://www.pinecone.io/learn/series/image-search/cnn/. Acessado em: 24 Junho, 2025.