



# INTELIGÊNCIA COMPUTACIONAL

## META 1 – ANÁLISE DO PROBLEMA E DESENVOLVIMENTO DE UM MODELO COM REDES CNN

OLEKSANDRA KUKHARSKA - 2020151174

### Conteúdo

<b>1. Capítulo 1: Descrição do Caso de Estudo e Objetivos do Problema .....</b>	<b>1</b>
<b>1.1. Contextualização do Problema.....</b>	<b>1</b>
<b>1.2. Descrição e Preparação do Dataset.....</b>	<b>1</b>
<b>2. Capítulo: Descrição da Implementação dos Algoritmos .....</b>	<b>2</b>
<b>2.1. Pré-Processamento dos Dados .....</b>	<b>2</b>
<b>2.2. Experimentos .....</b>	<b>2</b>
<b>2.2.1. Modelo 1 – Modelo original de 6 camadas .....</b>	<b>2</b>
<b>2.2.2. Modelo 2 - Prevenção de Overfitting .....</b>	<b>3</b>
<b>2.2.3. Modelo 3 - Aumento de Camadas Convulsionais .....</b>	<b>3</b>
<b>2.2.4. Modelo 4 – Diminuição de Camadas Densas.....</b>	<b>3</b>
<b>2.2.5. Modelo 5 – Alteração da Função de Otimização .....</b>	<b>4</b>
<b>3. Capítulo: Análise de Resultados.....</b>	<b>4</b>
<b>3.1. Modelo 1 – Modelo original de 6 camadas .....</b>	<b>4</b>
<b>3.2. Modelo 2 - Prevenção de Overfitting .....</b>	<b>4</b>
<b>3.3. Modelo 3 - Aumento de Camadas Convulsionais .....</b>	<b>5</b>
<b>3.4. Modelo 4 – Diminuição de Camadas Densas.....</b>	<b>5</b>
<b>3.5. Modelo 5 – Alteração da Função de Otimização .....</b>	<b>6</b>
<b>4. Capítulo: Conclusões.....</b>	<b>7</b>
<b>4.1. Síntese dos Resultados.....</b>	<b>7</b>
<b>Referências.....</b>	<b>9</b>

# 1. Capítulo 1: Descrição do Caso de Estudo e Objetivos do Problema

Neste Capítulo será descrito o problema de classificação de imagem escolhido, as características das amostras e o *Dataset* utilizado.

## 1.1. Contextualização do Problema











O objetivo deste estudo consiste em aplicar um algoritmo para classificação de imagem, avaliar os resultados, e fazer melhorias ao algoritmo de maneira a identificar corretamente as classes de imagens.

O algoritmo selecionado para o problema foi o *Convolutional Neural Network* (CNN), que será implementado em *Python*, no ambiente *JupyterLab*.

## 1.2. Descrição e Preparação do Dataset

O *Dataset* utilizado foi o Fruits-360, que consiste em imagens de frutas. Foram selecionadas 10 classes, com 70% das imagens para Treino, 10% para Validação e 20% para Teste. O número de imagens de Treino foi mantido, enquanto as imagens para Validação foram selecionadas aleatoriamente utilizando a função *random()*, para representar 10% do *Dataset*, o número de imagens do conjunto de Teste original foi ajustado para representar 20% do total de amostras. O *Dataset* não é balanceado. A amostra total contém 28183 imagens de 100x100, em RGB, conforme apresentado na Tabela 1.

Tabela 1 Dataset original

				
Apple	Banana	Blueberry	Cherry	Fig
Treino: 6632 Validação: 947 Teste: 1895	Treino: 1251 Validação: 179 Teste: 358	Treino: 404 Validação: 58 Teste: 116	Treino: 3014 Validação: 431 Teste: 861	Treino: 614 Validação: 88 Teste: 176
				
Lemon	Orange	Pear	Pineapple	Strawberry
Treino: 859 Validação: 123 Teste: 246	Treino: 419 Validação: 60 Teste: 120	Treino: 4595 Validação: 657 Teste: 1313	Treino: 860 Validação: 123 Teste: 246	Treino: 1076 Validação: 154 Teste: 308

## 2. Capítulo: Descrição da Implementação dos Algoritmos

No Capítulo 2, serão descritos diferentes experimentos realizados com o algoritmo CNN, abrangendo diferentes configurações da arquitetura da rede.

### 2.1. Pré-Processamento dos Dados

Para resolver o problema do desbalanceamento do *Dataset*, optei por utilizar as técnicas *Oversampling* e *Undersampling*, para duplicar e eliminar imagens, respetivamente. Desta forma, o *Dataset* utilizado para o problema possui no total 43060 imagens.

No caso das classes minoritárias, a duplicação de amostras pode provocar falta de variedade, o que prejudica a performance para classificação, dessa forma é aplicada a técnica de *Data Augmentation*, para aumentar a diversidade de imagens:

**Output:**

```
#Criar gerador de dados com aumento para treino
train_datagen = ImageDataGenerator(
    rescale=1.0/255, # Normalização
    rotation_range=40, # Rotação aleatória de até 40 graus
    width_shift_range=0.2, # Deslocamento horizontal aleatório de até 20%
    height_shift_range=0.2, # Deslocamento vertical aleatório de até 20%
    shear_range=0.2, # Aplicar cisalhamento
    zoom_range=0.2, # Zoom aleatório
    horizontal_flip=True, # Inversão horizontal
    fill_mode='nearest' # Preencher pixels vazios
)
```

Em contraste, o gerador de validação e teste foi configurado sem aumentos, focando apenas na normalização das imagens. garantindo que a avaliação do desempenho do modelo seja realizada num *Dataset* que reflete as condições reais, sem introduzir variações artificiais.

### 2.2. Experimentos

São realizados 6 experimentos para o problema de classificação de imagem, nos quais serão feitas alterações aos Hiper parâmetros do algoritmo. Todos os modelos têm epoch=30 e batch size=32.

#### 2.2.1. Modelo 1 – Modelo original de 6 camadas

Como sugerido na proposta, o modelo inicial é um CNN de 6 camadas, é utilizado o otimizador Adam, por ser considerado “padrão”.

O Modelo 1 consiste em duas camadas Convolucionais que aplicam filtros para extrair características básicas, como bordas, além de características mais complexas, como formas. Após cada camada Convolutiva, uma camada de

*Pooling* é aplicada para reduzir a dimensionalidade dos dados, e preservar informações relevantes, o modelo tem assim a seguinte estrutura:

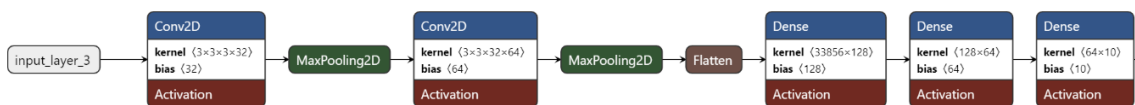


Figura 1 Camadas do Modelo 1

Fonte: [netron.app](http://netron.app)

### 2.2.2. Modelo 2 - Prevenção de Overfitting

Neste experimento são feitas melhorias ao Modelo 1, adicionando os Hiperparâmetros para evitar *Overfitting*, e aprender a generalizar as previsões para novas imagens, em vez de memorizar os dados de Treino. Os parâmetros adicionados são:

- *Batch Normalization*: Aplicadas às camadas Convolucionais;
- *Dropout*: Desligar aleatoriamente 50% dos neurônios da camada Densa;
- *Early-Stopping*: Interrompe o treino automaticamente se o desempenho na validação parar de melhorar por um certo número de *epoch*.

### 2.2.3. Modelo 3 - Aumento de Camadas Convulsionais

Após regular o *Overfitting*, no Modelo 3, são adicionadas duas camadas Convolucionais, uma com 128 filtros e outra com 256 filtros, para o modelo aprender características mais complexas das imagens.

O *Batch Normalization* e o *Pooling* são mantidos após cada camada Convolutiva para reduzir a dimensionalidade das representações.

A primeira camada Densa foi aumentada para 256 unidades, com a intenção de ajudar o modelo a capturar mais informações antes da camada de saída.

O Modelo 3, tem a seguinte estrutura:

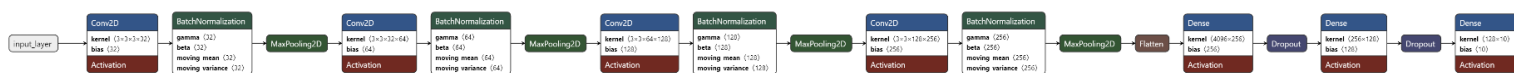


Figura 2 Camadas do Modelo 3

Fonte: [netron.app](http://netron.app)

### 2.2.4. Modelo 4 – Diminuição de Camadas Densas

Pelos resultados obtidos nos Modelos anteriores, e com a Prevenção de *Overfitting*, optei por reduzir o número de camadas Densas, já que os modelos apresentavam em determinadas *epoch*, uma *Accuracy* de Treino perfeita, sendo indicador de que o modelo não está a generalizar corretamente, mas a memorizar dados de Treino.

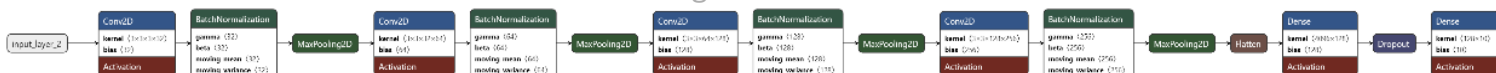


Figura 3 Camadas do Modelo 4

Fonte: [netron.app](https://netron.app)

## 2.2.5. Modelo 5 – Alteração da Função de Otimização

Com base no Modelo 3, que mostrou resultados mais promissores que os demais, irá ser feita alteração ao tipo de otimizador, passando a utilizar o *SGD* pela sua simplicidade e eficácia em problemas de otimização. Além disso, o *RMSprop* será considerado, pois ajusta a *Learning Rate* de forma adaptativa.

## 3. Capítulo: Análise de Resultados

Após a implementação de cada modelo, são extraídos os valores das métricas, e consequentes gráficos para visualizar o desempenho de cada modelo ao longo das *epoch*, que serão de seguida analisados.

### 3.1. Modelo 1 – Modelo original de 6 camadas

O Modelo 1, para o Treino, obteve uma *Loss* extremamente baixa, de 0.0004, e uma *Accuracy* de 100%, o que sugere que o modelo ajustou-se perfeitamente aos dados de Treino, o que pode ser um sinal de *Overfitting*, como se apresenta na Figura 4.

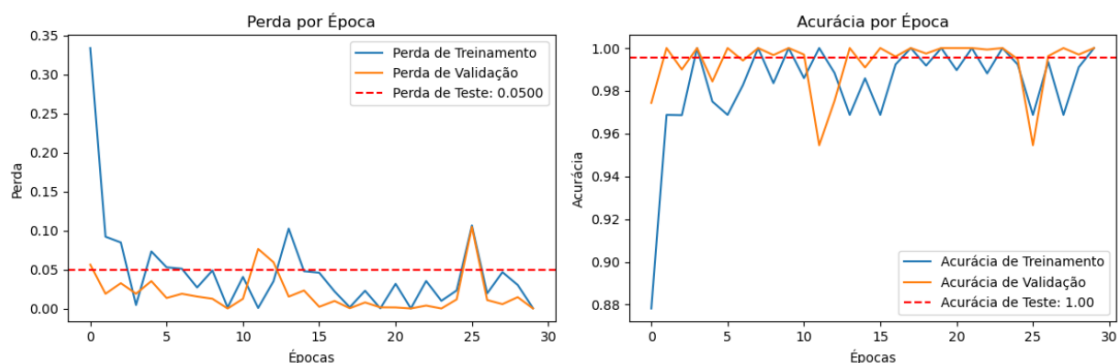


Figura 4 Loss e Accuracy do Modelo 1

Para Validação, o modelo alcançou uma *Loss* de 0.0109 e *Accuracy* de 99.77%. O modelo acertou 99,57% das classificações nos dados de Teste. Apesar da performance elevada, o modelo não me parece que se ajuste bem num cenário de dados mais diversos.

### 3.2. Modelo 2 - Prevenção de Overfitting

O modelo apresenta uma *Accuracy* de 96,05% nos dados de Teste e *Loss* de Validação e Teste significativamente mais baixas. Como se observa na Figura 5, há oscilação nos dados durante o Treino, o que indica que o modelo está com dificuldades de se adaptar adequadamente a novas informações, entre a *epoch* 10

e 15, há indícios de *Overfitting*, já que começa a haver uma separação dos dados de Treino e de Validação.

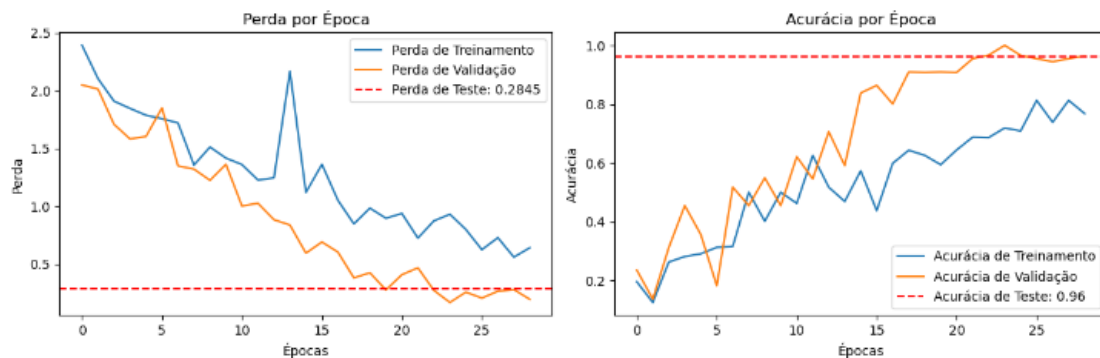


Figura 5 Loss e Accuracy do Modelo 2

### 3.3. Modelo 3 - Aumento de Camadas Convulsionais

O modelo apresenta uma *Accuracy* de Teste de 98.76% e uma *Loss* de apenas 0.0343, o que indica uma forte capacidade de generalização para dados não vistos. Existe um elevado equilíbrio da taxa de verdadeiros positivos e negativos.

Como se observa na figura seguinte, foram observados picos de *Accuracy* e *Loss* na Validação que chegaram a valores superiores a 0,9 e, em seguida, caíram para 0, indicando que o modelo pode estar a enfrentar dificuldades em se adaptar a certas variações nos dados, ou seja, o modelo aprende bem com os dados de Treino, mas não generaliza bem com os dados de Validação. Algumas *epoch* para Treino apresenta valores perfeitos, de 1.0000, o que indica que poderá haver algum *Overfitting*.

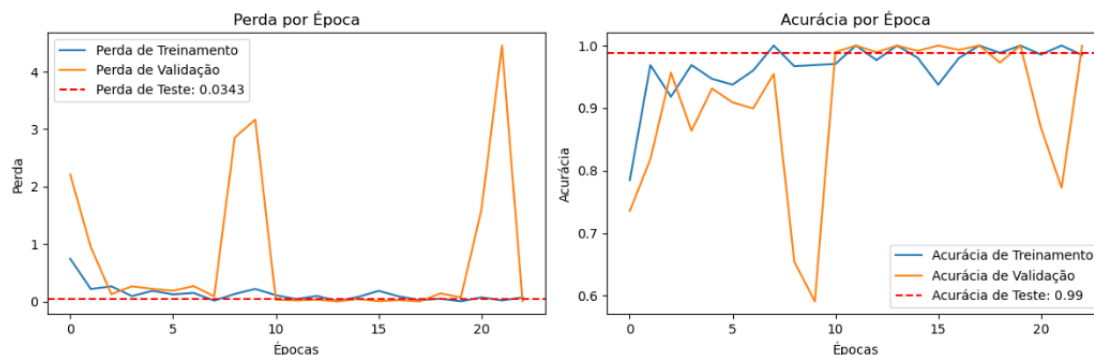


Figura 6 Loss e Accuracy do Modelo 3

### 3.4. Modelo 4 – Diminuição de Camadas Densas

Com a diminuição do número de camadas Densas, com *Accuracy* de 97,42% no Teste e 98,35% na Validação, evidencia a capacidade do modelo de generalizar bem para novos dados. *Loss* de teste foi de 0,1442, o que indica que o modelo conseguiu minimizar os erros de previsão. Esse ajuste nas camadas Densas contribuiu para

uma estrutura mais eficiente, evita complexidade excessiva, o que resultou em um modelo robusto e preciso.

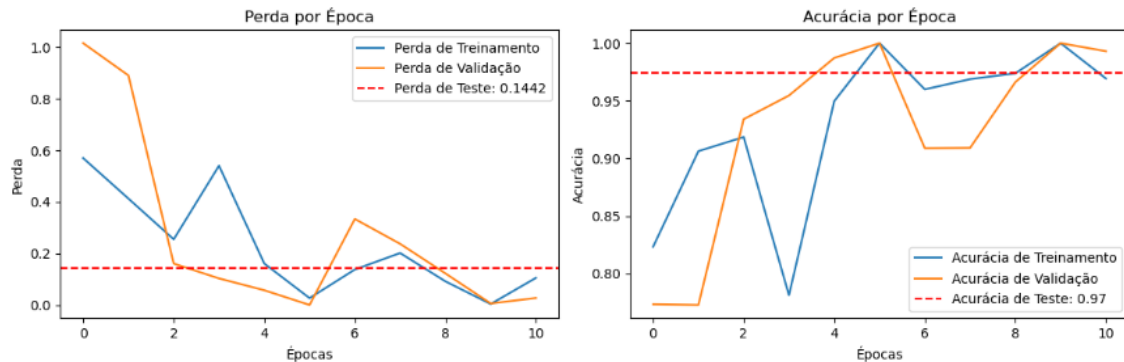


Figura 7 Loss e Accuracy do Modelo 4

### 3.5. Modelo 5 – Alteração da Função de Otimização

O Modelo com o otimizador SGD, dá indícios de *Overfitting*, uma vez que a curva da Validação atinge valor perfeito em várias *epoch*. E com o otimizador RMSprop há uma diminuição nos valores das métricas.

#### 3.5.1. Uso do otimizador SGD

O modelo demonstrou um desempenho notável nas diferentes etapas do processo de Treino e Validação. No Teste, a *Loss* foi de 0.0406, resultando numa *Accuracy* de 99.19%. Essas métricas indicam que o modelo consegue classificar com alta precisão os dados não vistos, mantendo um equilíbrio entre a taxa de falsos positivos e negativos. Na Validação, a *Loss* foi significativamente baixa, de 0.0118, e a *Accuracy* alcançou 99.63%, dando sinais de *Overfitting*. No Treino, a *Loss* foi de 0.0722 e a *Accuracy* de 98.03%, indicando que o modelo aprendeu bem a partir dos dados de Treino, embora exista uma leve discrepância entre as métricas de Treino e Validação.

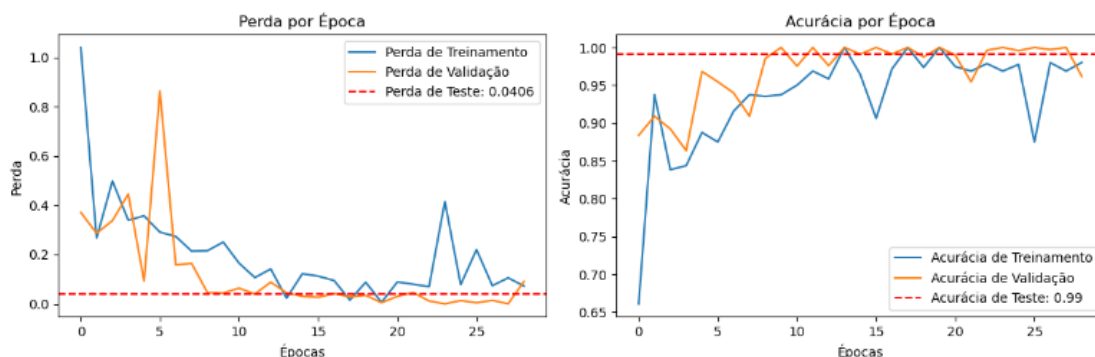


Figura 8 Loss e Accuracy do Modelo 5 com SGD

#### 3.5.2. Uso do otimizador RMSprop

No Teste, o modelo obteve uma *Accuracy* de 95.31% e uma *Loss* de 0.1735, o que demonstra uma boa capacidade de classificação de dados não vistos. Durante a validação, o modelo apresentou uma *Accuracy* de 95.06% e uma *Loss* de 0.1765, apesar de algumas flutuações, ele generaliza bem para dados que não foram utilizados durante o Treino. No Treino há uma *Accuracy* de 87.50% e uma *Loss* de 0.5431, indicando que o modelo ainda poderia melhorar na sua capacidade de aprendizagem.

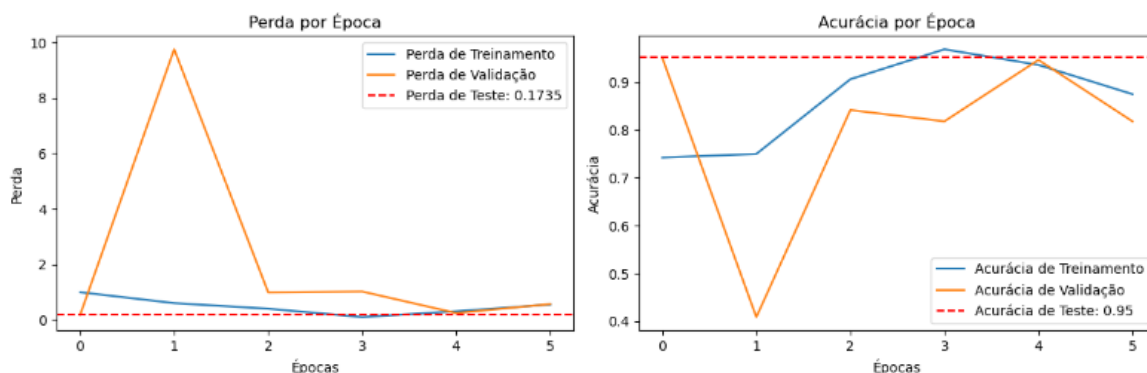


Figura 9 Loss e Accuracy do Modelo 5 com RMSprop

## 4. Capítulo: Conclusões

Neste Capítulo são mencionadas as considerações finais sobre os experimentos realizados, nomeadamente uma comparação entre os 6 Modelos.

### 4.1. Síntese dos Resultados

A Tabela seguinte mostra os resultados obtidos com todos os Modelos.

Tabela 2 Resultados dos Modelos

Experimento	Modelo 1 (Original)	Modelo 2 (Previne Overfitting)	Modelo 3 (Aumentar Camada Conv)	Modelo 4 (Diminuição Camada Densa)	Modelo 5 (Alterar Função Otimização)	
					SGD	RMSprop
Train Loss	0.0004	0.6398	0.0669	0.1054	0.0722	0.5431
Train Accuracy	1.0000	0.7678	0.9848	0.9693	0.9803	0.8750
Val Loss	0.0109	0.2690	0.0209	0.0675	0.0118	0.1765
Val Accuracy	0.9977	0.9671	0.9926	0.9835	0.9963	0.9506
Test Loss	0.0500	0.2845	0.0343	0.1442	0.0406	0.1735
Test Accuracy	0.9957	0.9605	0.9876	0.9742	0.9919	0.9531
Precision	0.9958	0.9614	0.9884	0.9756	0.9919	0.9530
Recall	0.9957	0.9605	0.9876	0.9742	0.9919	0.9531
F1 Score	0.9957	0.9608	0.9875	0.9742	0.9919	0.9514



Como se observa, o Modelo 1, apresenta *Overfitting*, consegue uma *Train Accuracy* de 1.0000. Após uso de técnicas como *Dropout*, *Early Stopping* e *Batch Normalization*, o Modelo 2, apresenta valores mais robustos, em que não se verifica que o Modelo esteja a “decorar” as classificações das imagens de Treino, tendo apresentado um valor significativo de falsos positivos para a classe Pear que foi confundida com a classe Apple.

No Modelo 3, quando se aumenta o número de camadas Convolucionais, os valores são ligeiramente melhores ao Modelo anterior, tendo até diminuído o número de falsos positivos da classe Pear.

O Modelo 4, apesar de diminuir falsos positivos na classe Pear, acaba por criar mais falsos positivos na classe Apple.

Como o Modelo 3 apresenta melhores valores que o Modelo 4, foi baseado nele que se fizeram os experimentos ao Modelo 5. Comparando ambos os optimizadores, o SGD fornece uma previsão mais precisas, no entanto, o optimizador Adam consegue prever maior número de classes.

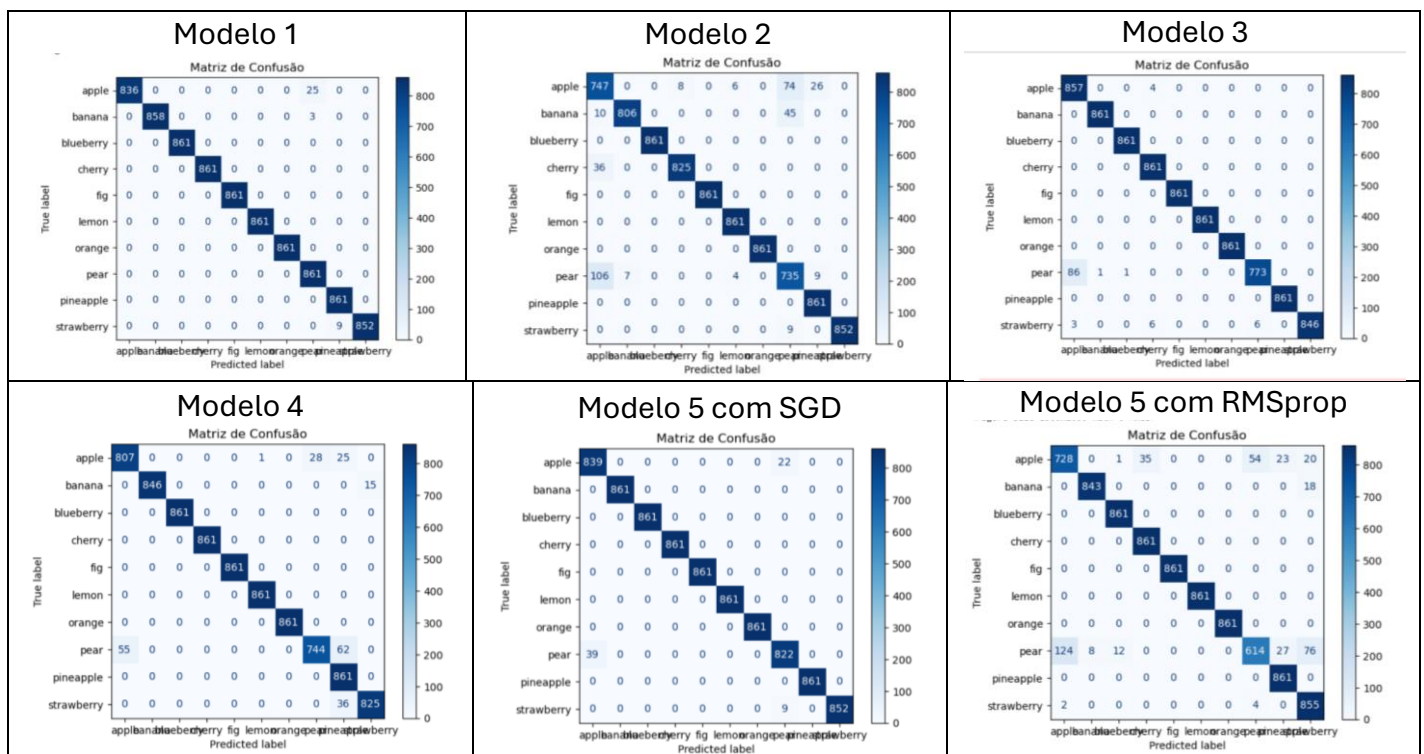


Figura 10 Matizes de Confusão para os 6 Modelos

## Referências

Almalik, A. (2023, 23 de julho). \*Convolutional neural networks for image classifying using Python\*. Medium. <https://medium.com/@afghanalmalik2404/convolutional-neural-networks-for-image-classifying-using-python-3968d7b1f17b>

DataCamp. (n.d.). *CNN with TensorFlow in Python: A step-by-step guide*. <https://www.datacamp.com/tutorial/cnn-tensorflow-python>

Kumar, A. (2020, January 21). Learning rate schedules and adaptive learning rate methods for deep learning. Towards Data Science. <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>

OpenAI. (2024). \*ChatGPT (versão 4)\*. <https://chat.openai.com>

TensorFlow. (n.d.). *Convolutional neural networks (CNNs)*. TensorFlow. <https://www.tensorflow.org/tutorials/images/cnn?hl=pt-br>

Walraven, B. C. (2020, 3 de abril). *Boost your CNN with the Keras ImageDataGenerator*. Medium. <https://medium.com/@bcwalraven/boost-your-cnn-with-the-keras-imagedatagenerator-99b1ef262f47>