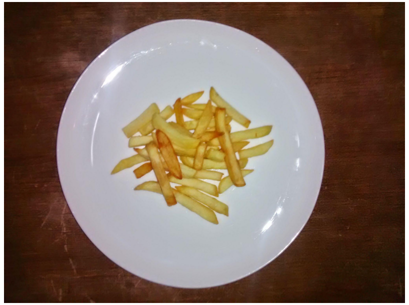
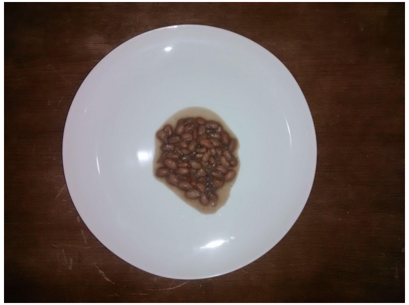
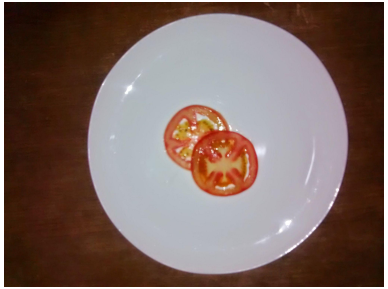
Classificação de Alimentos com CNN

Da Base à Otimização

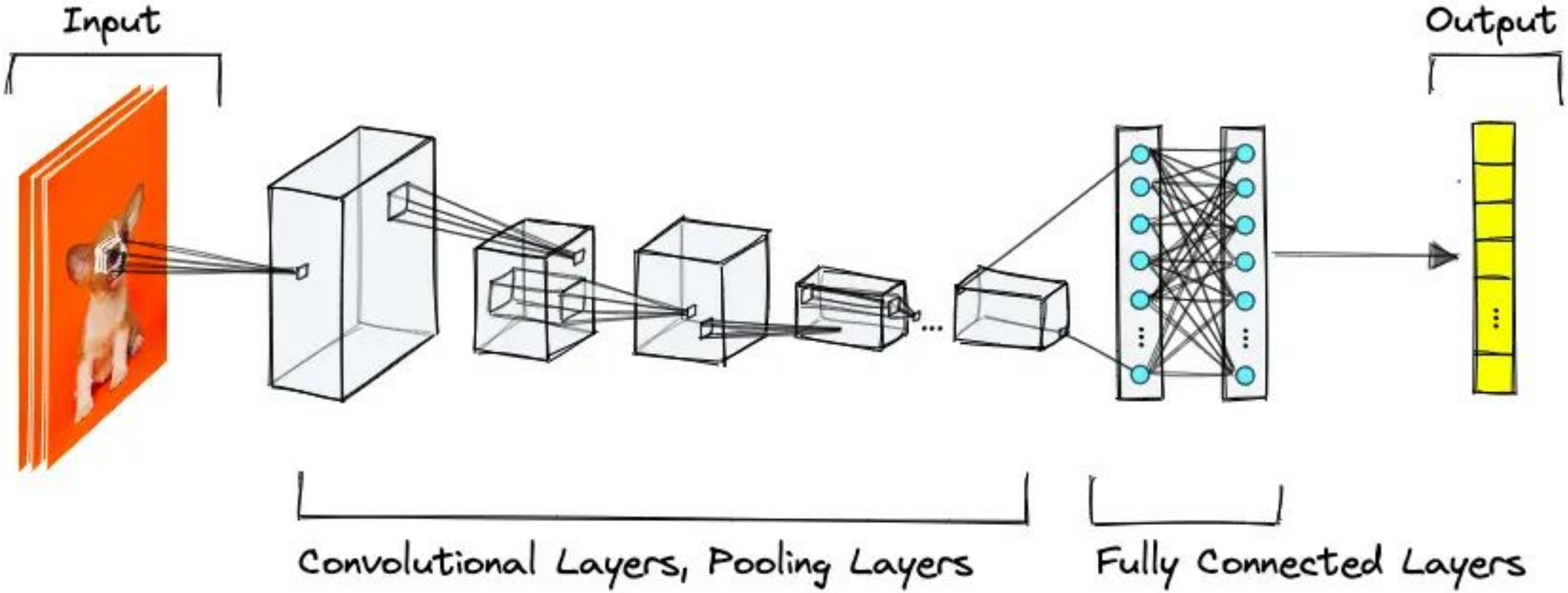
Gustavo Adame Domingues - 2280515

Objetivo

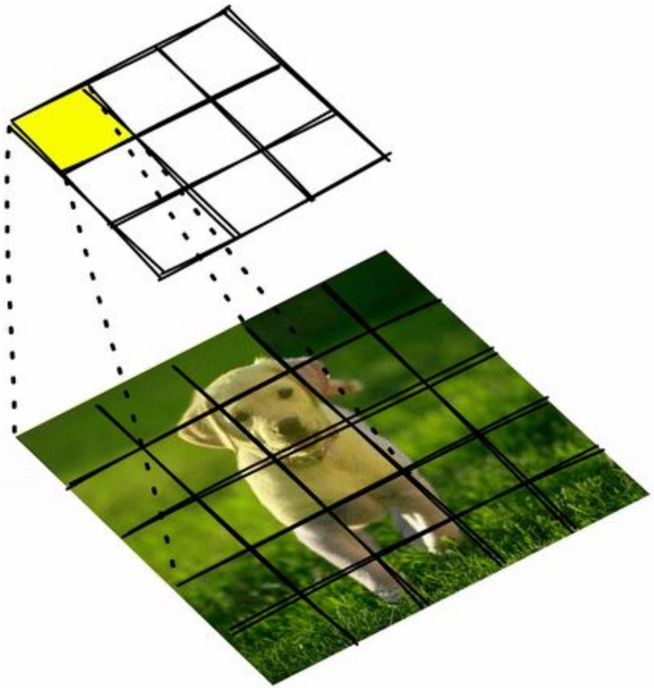
- Desenvolver, treinar e otimizar um modelo de Rede Neural Convolucional (CNN) capaz de classificar 16 tipos diferentes de alimentos com alta acurácia.

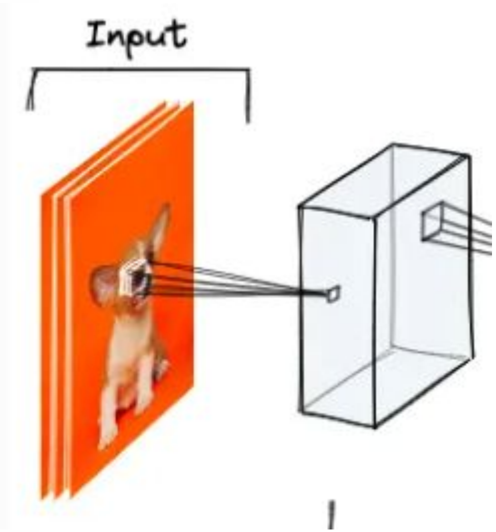
   

O que é uma CNN?

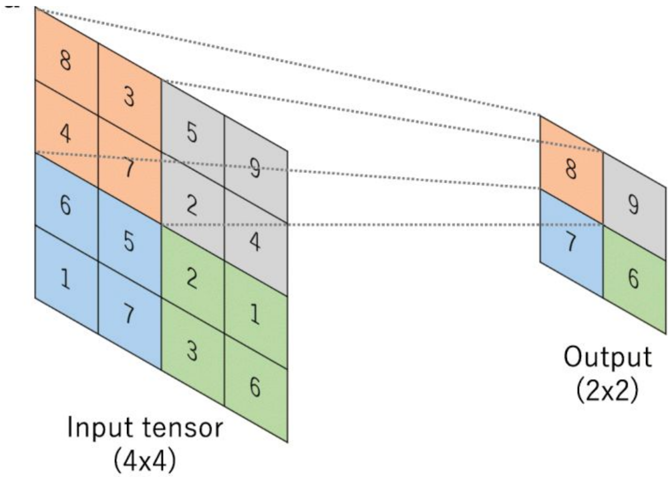


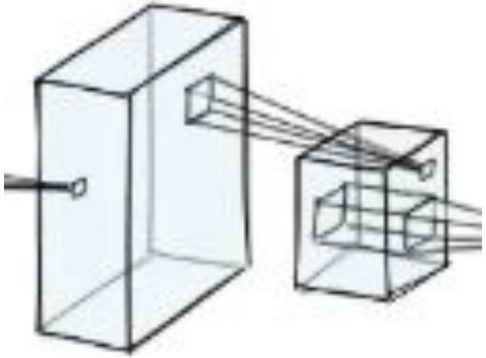
Camadas convolucionais

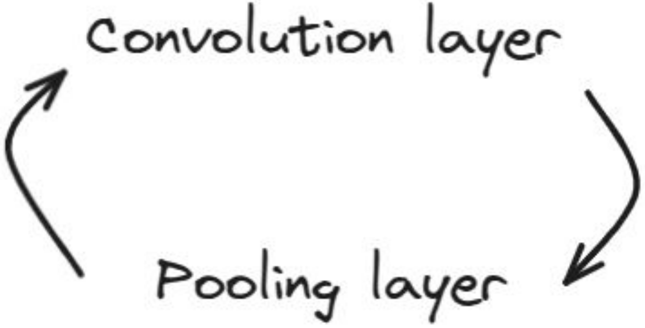


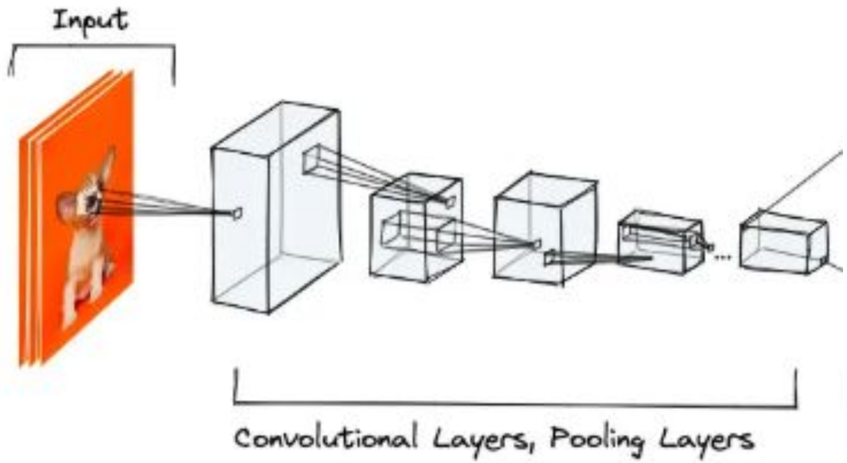


Camada de pooling



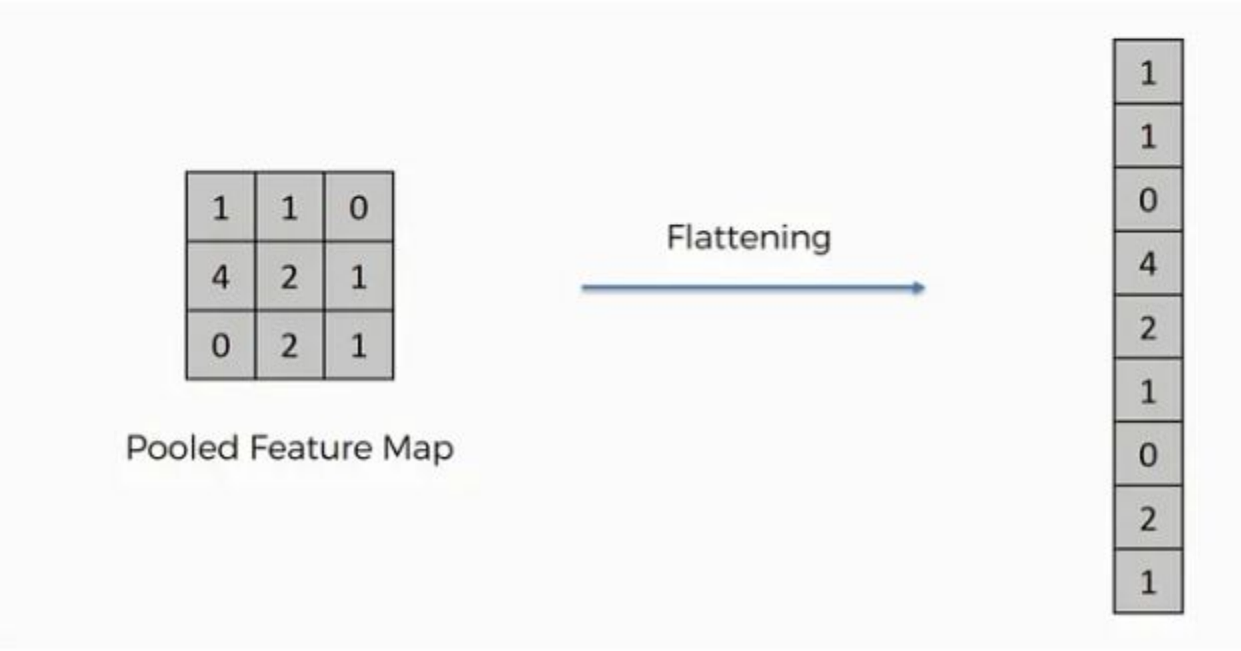



Convolution layer
Pooling layer

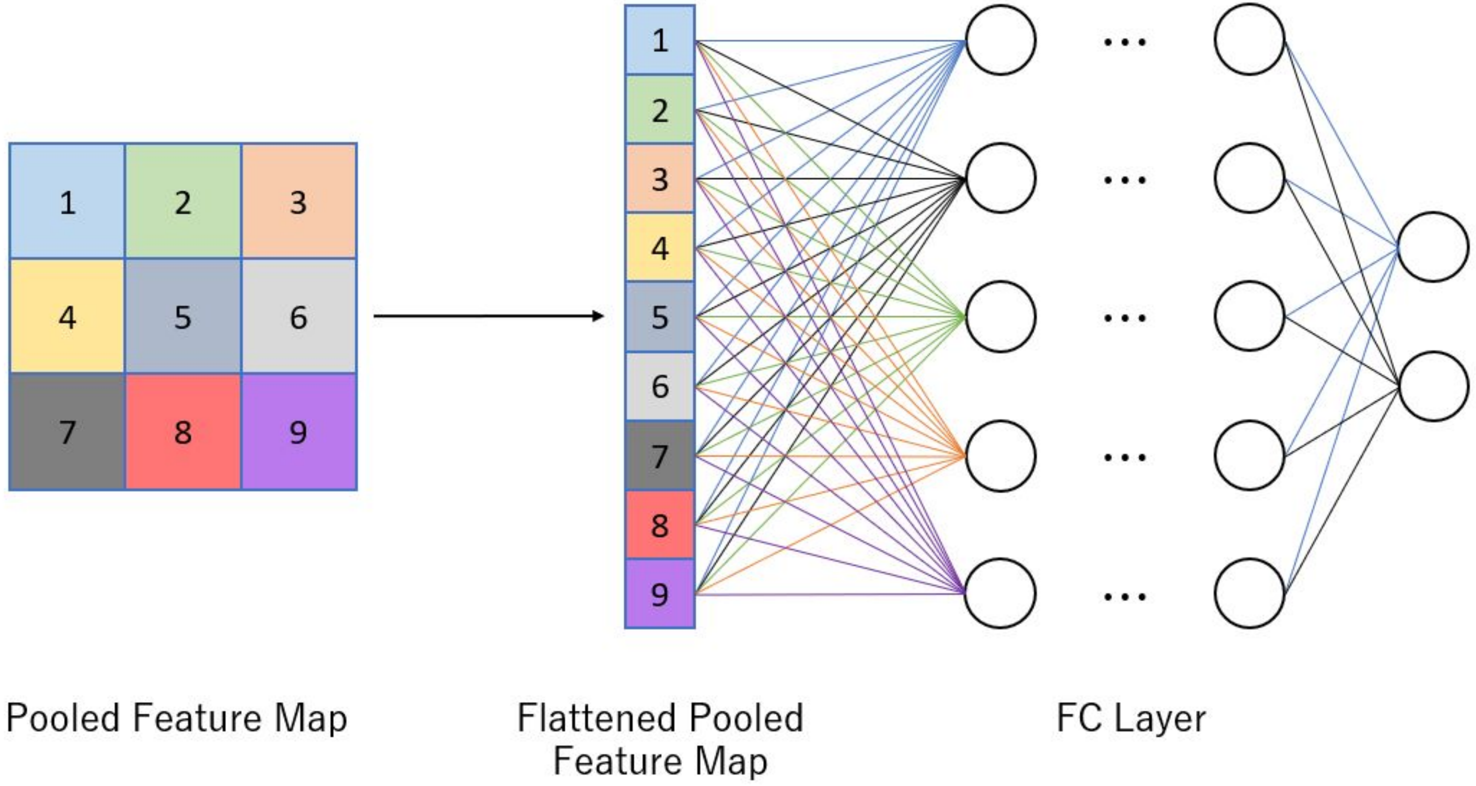

Input
Convolutional Layers, Pooling Layers

```python
import tensorflow as tf
from tensorflow.keras import layers, models # type: ignore

# --- ARQUITETURA DA CNN ---

# Inicia um modelo sequencial, onde as camadas são empilhadas uma após a outra
model = models.Sequential(name='CNN_Alimentos_Parte1')

# --- BLOCO 1 ---
# Camada Convolucional inicial para aprender 32 padrões (filtros) simples.
# O input_shape deve corresponder ao target_size dos seus geradores.
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3), padding='same', name='conv1_1'))
# Camada de Pooling para reduzir as dimensões pela metade e reter as características mais fortes.
model.add(layers.MaxPooling2D((2, 2), name='pool1'))

# --- BLOCO 2 ---
# Aumentamos para 64 filtros para que a rede possa aprender padrões mais complexos
# a partir das características simples da camada anterior.
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same' ,name='conv2_1'))
model.add(layers.MaxPooling2D((2, 2), name='pool2'))

# --- BLOCO 3 ---
# Mais uma camada com 128 filtros para aprender abstrações de nível ainda mais alto.
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same', name='conv3_1'))
model.add(layers.MaxPooling2D((2, 2), name='pool3'))
```

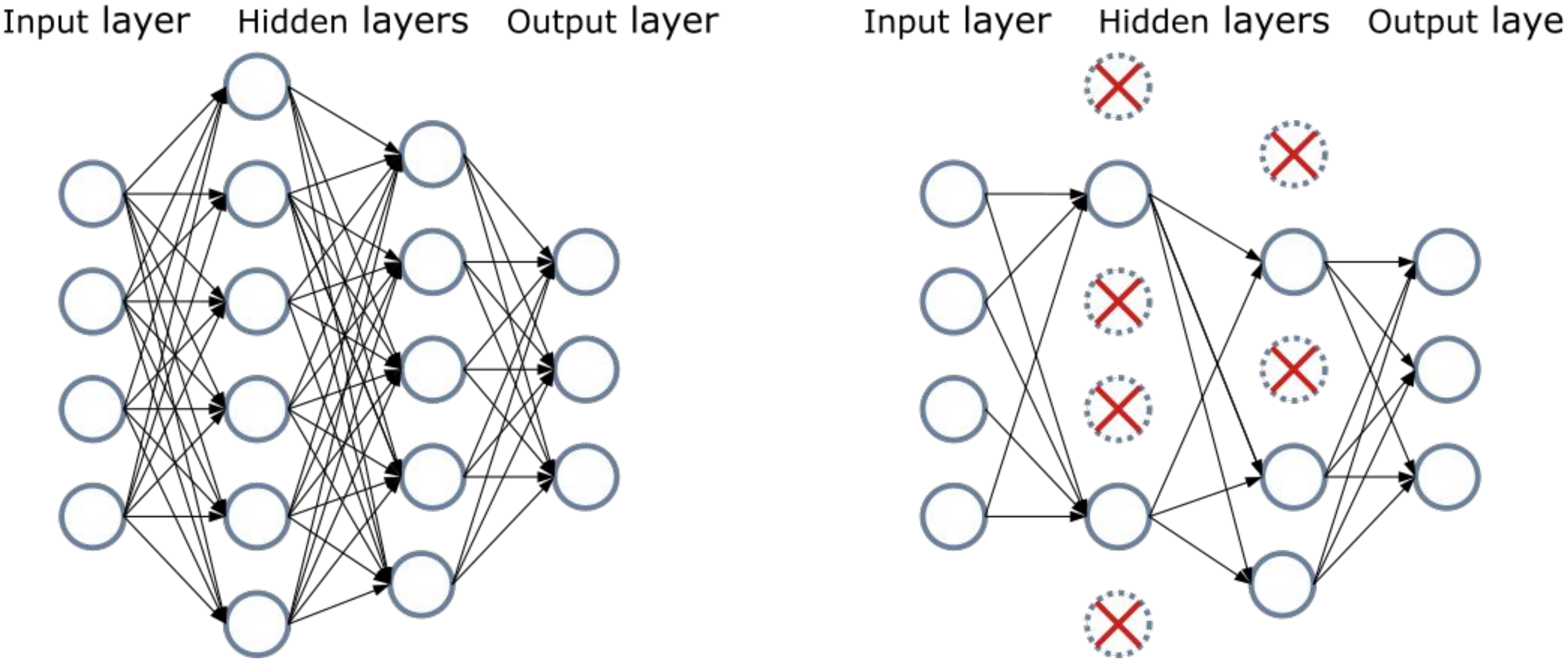
Camada flatten



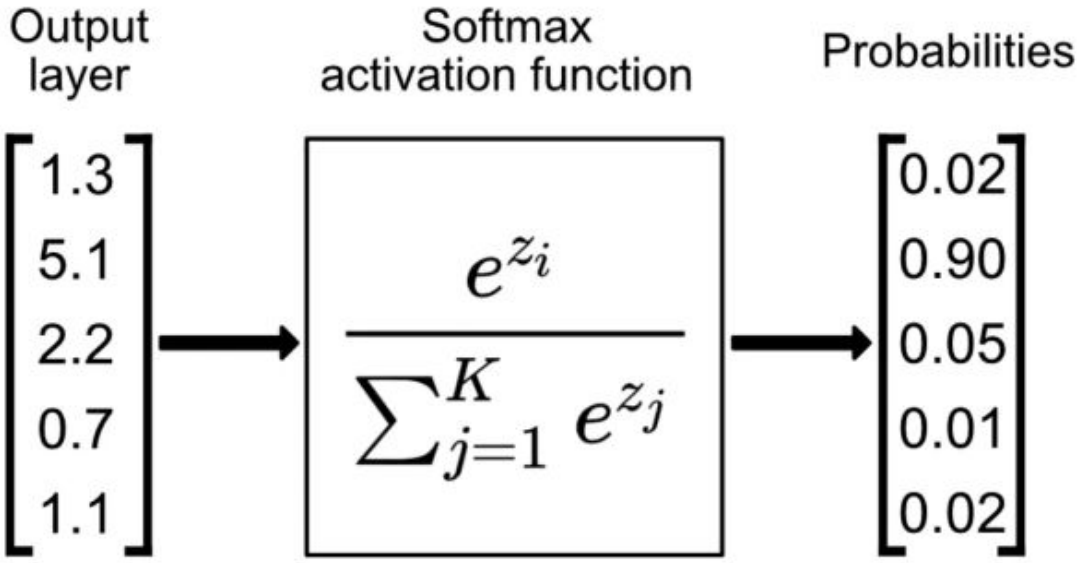
Camada totalmente conectada

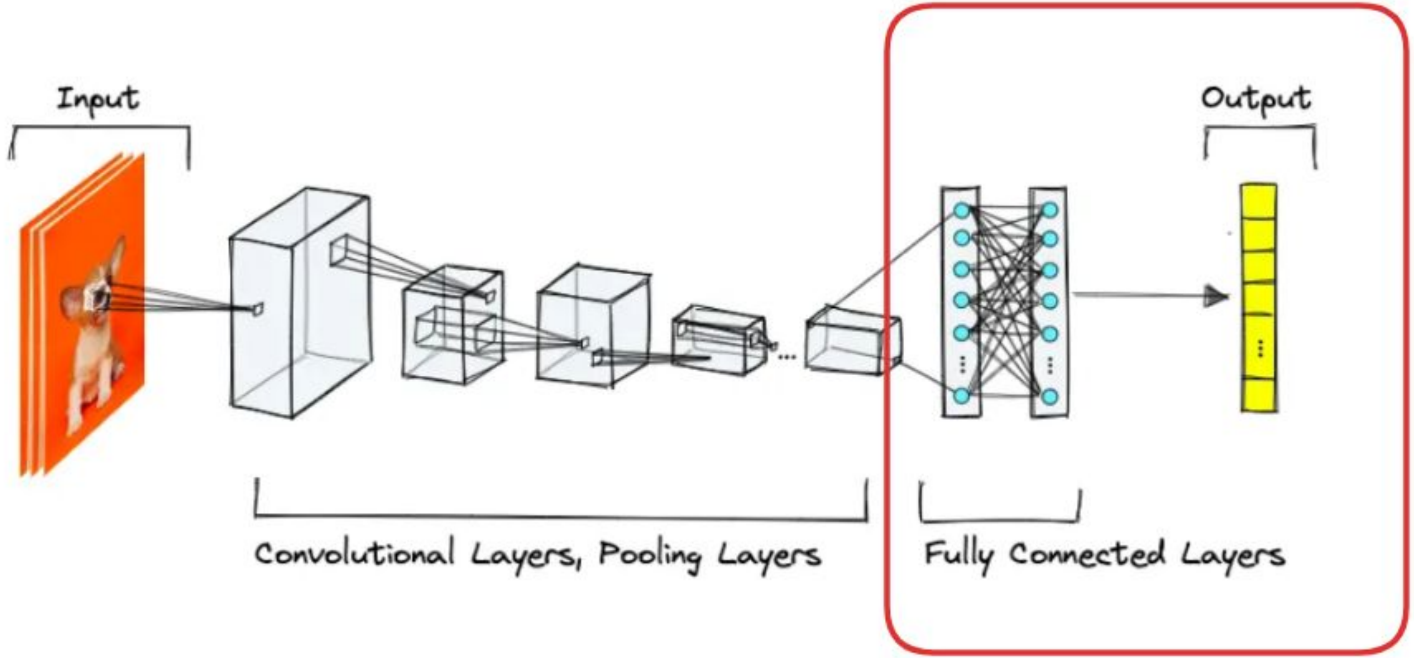


Camada de dropout



Camada Softmax




Input
Output
Convolutional Layers, Pooling Layers
Fully Connected Layers

```python
# --- CLASSIFICADOR (MLP) ---

# 1. Achatamos (Flatten) o mapa de características 3D para um vetor 1D
# para que ele possa ser processado pelas camadas Densas.
model.add(layers.Flatten(name='flatten'))

# 2. Camada Densa (Totalmente Conectada) com 256 neurônios para aprender
# a combinar todas as características extraídas.
model.add(layers.Dense(256, activation='relu', name='dense1'))

# 3. Camada de Dropout para combater o overfitting. Ele "desliga" aleatoriamente
# 50% dos neurônios durante o treino para forçar a rede a não depender de um
# único caminho, tornando-a mais robusta.
model.add(layers.Dropout(0.5, name='dropout'))

# 4. Camada de Saída. O número de neurônios deve ser igual ao número de classes.
# A ativação 'softmax' transforma a saída em probabilidades para cada classe.
model.add(layers.Dense(16, activation='softmax', name='output'))
```

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

Compilação do modelo base

```
print("Compilando o modelo...")
model.compile(
    optimizer='sgd',                      # Otimizador Stochastic Gradient Descent (passos fixos)
    loss='categorical_crossentropy',      # Função de perda para classificação multiclasse
    metrics=['accuracy']                  # Métrica para monitorar o desempenho
)
print("Modelo compilado com sucesso!")
```

✓ 0.0s

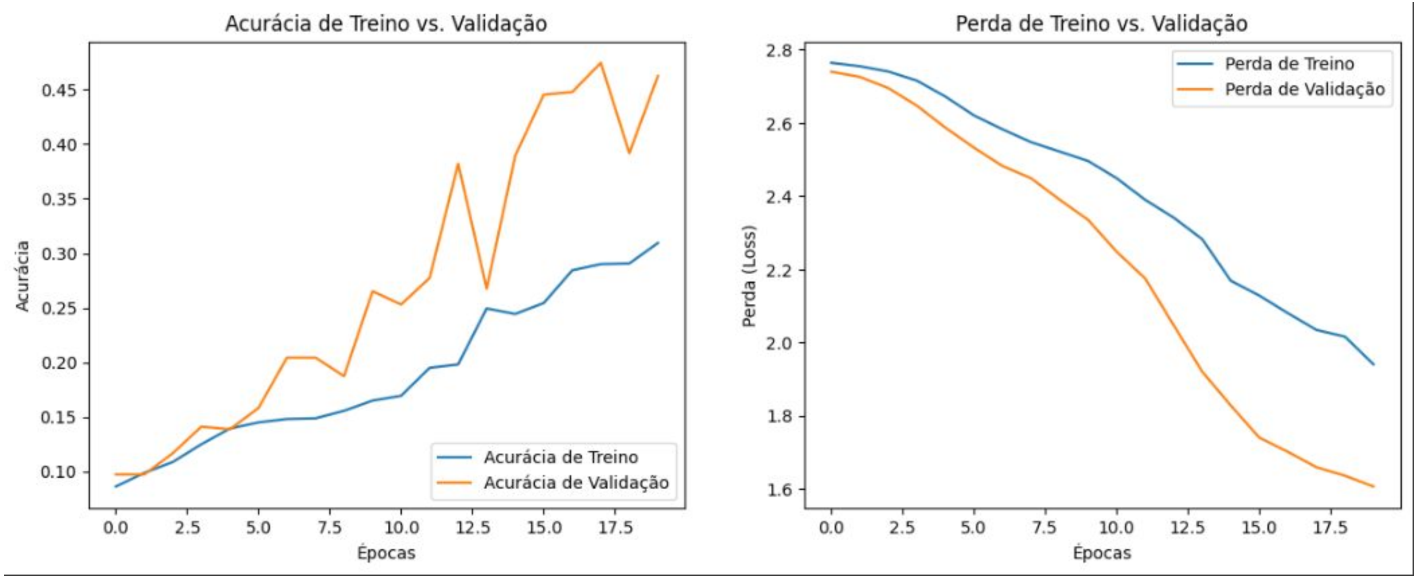
Treinamento do modelo base

```python
EPOCHS = 20

print(f"Iniciando o treinamento por {EPOCHS} épocas...")

history = model.fit(
    train_generator,           # Fornece os lotes de dados de treino (com augmentation)
    epochs=EPOCHS,             # Número de ciclos de treinamento
    validation_data=validation_generator, # Fornece os dados de validação para teste ao fim de cada época
    verbose=1                  # Mostra uma barra de progresso (1=sim, 0=não)
)

print("Treinamento concluído!")
```

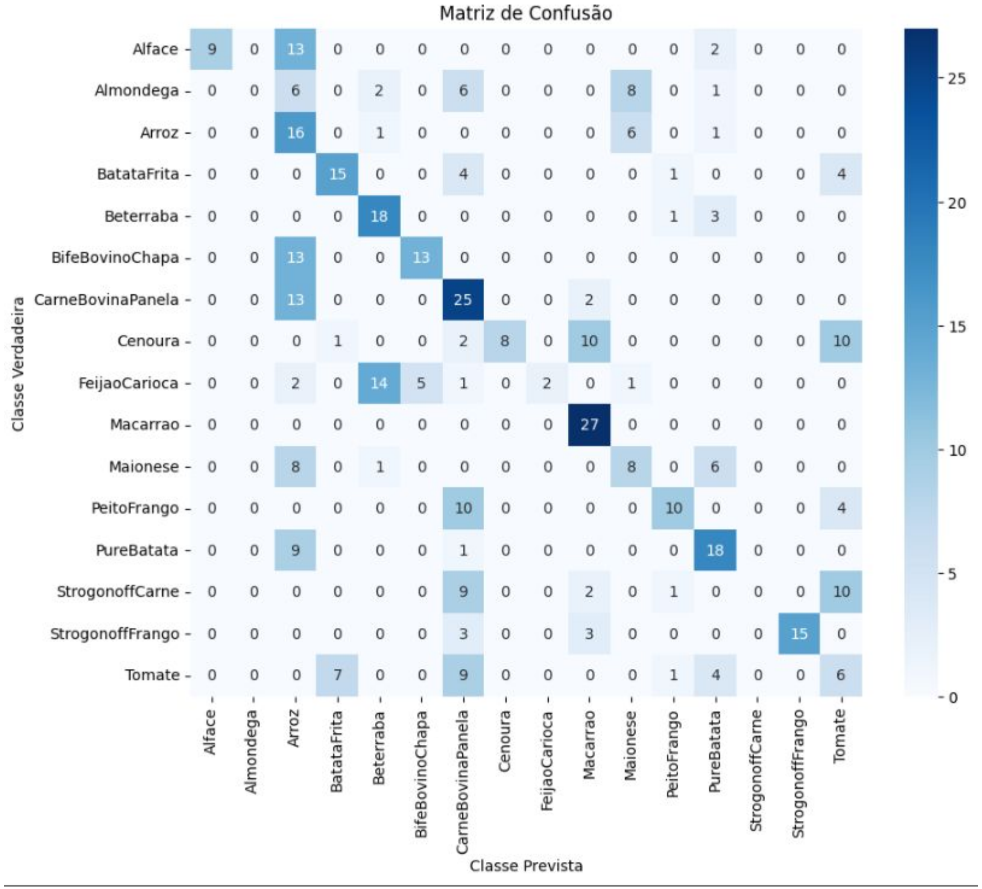
Treinamento do modelo base

```
Iniciando o treinamento por 20 épocas...
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your
  self._warn_if_super_not_called()
Epoch 1/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 131s 3s/step - accuracy: 0.0754 - loss: 2.7738 - val_accuracy: 0.0973 - val_loss: 2.7401
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.0850 - loss: 2.7538 - val_accuracy: 0.0973 - val_loss: 2.7259
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 133s 3s/step - accuracy: 0.1023 - loss: 2.7417 - val_accuracy: 0.1168 - val_loss: 2.6953
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.1138 - loss: 2.7262 - val_accuracy: 0.1411 - val_loss: 2.6472
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 131s 3s/step - accuracy: 0.1358 - loss: 2.6840 - val_accuracy: 0.1387 - val_loss: 2.5871
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 133s 3s/step - accuracy: 0.1342 - loss: 2.6303 - val_accuracy: 0.1582 - val_loss: 2.5326
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 132s 3s/step - accuracy: 0.1511 - loss: 2.5949 - val_accuracy: 0.2044 - val_loss: 2.4824
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.1502 - loss: 2.5273 - val_accuracy: 0.2044 - val_loss: 2.4491
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.1603 - loss: 2.5248 - val_accuracy: 0.1873 - val_loss: 2.3906
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 149s 3s/step - accuracy: 0.1686 - loss: 2.4890 - val_accuracy: 0.2652 - val_loss: 2.3352
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 133s 3s/step - accuracy: 0.1792 - loss: 2.4502 - val_accuracy: 0.2530 - val_loss: 2.2484
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.1811 - loss: 2.3855 - val_accuracy: 0.2774 - val_loss: 2.1753
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 126s 3s/step - accuracy: 0.2013 - loss: 2.3429 - val_accuracy: 0.3820 - val_loss: 2.0474
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 128s 3s/step - accuracy: 0.2476 - loss: 2.3099 - val_accuracy: 0.2676 - val_loss: 1.9206
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.2438 - loss: 2.1717 - val_accuracy: 0.3893 - val_loss: 1.8282
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.2480 - loss: 2.1390 - val_accuracy: 0.4453 - val_loss: 1.7406
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.2940 - loss: 2.0989 - val_accuracy: 0.4477 - val_loss: 1.7018
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 129s 3s/step - accuracy: 0.2894 - loss: 2.0372 - val_accuracy: 0.4745 - val_loss: 1.6597
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.2754 - loss: 2.0504 - val_accuracy: 0.3917 - val_loss: 1.6366
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.2979 - loss: 1.9555 - val_accuracy: 0.4623 - val_loss: 1.6070
Treinamento concluído!
```

Resultado - Modelo Base



Resultado - Modelo Base



Resultado - 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

Otimização do modelo base

Adição de mais uma camada de filtros

```
# --- 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'))

# --- FIM DA ARQUITETURA ---
model.summary()
```

Adição de Callbacks

ModelCheckpoint

```python
# 1. Salva o melhor modelo encontrado com base na menor perda de validação
model_checkpoint = ModelCheckpoint(
    filepath='modelo_otimizado.h5',  # Nome do arquivo para salvar o melhor modelo
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)
```

EarlyStopping

```
# 2. Para o treinamento se não houver melhora na perda de validação por 5 épocas
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True, # Restaura os pesos da melhor época encontrada
    verbose=1
)
```

EarlyStopping

```
# 2. Para o treinamento se não houver melhora na perda de validação por 5 épocas
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True, # Restaura os pesos da melhor época encontrada
    verbose=1
)
```

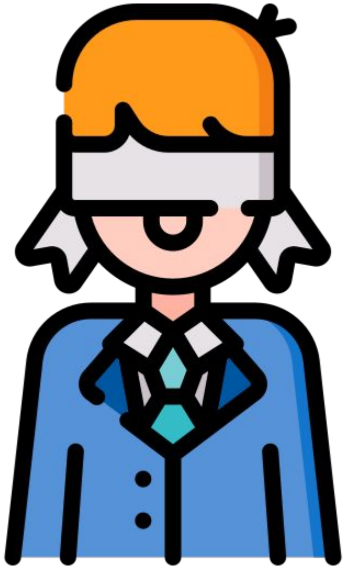
ReduceLROnPlateau

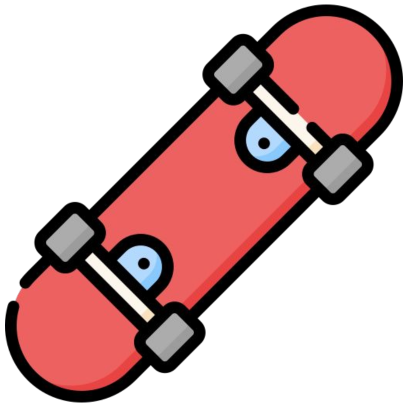
```python
# 3. Reduz a taxa de aprendizado se a perda de validação não melhorar por 5 épocas
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2, # Reduz o LR por um fator de 5 (1/5 = 0.2)
    patience=5,
    verbose=1
)
```

Compilação do novo modelo

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ]
)
```

Otimizador ADAM - Adaptive moment estimation

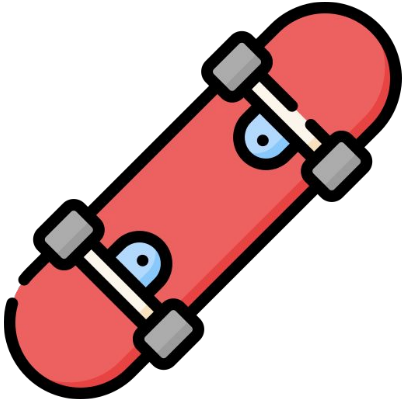




Momento

RMSprop

Otimizador ADAM - Adaptive moment estimation



Momento

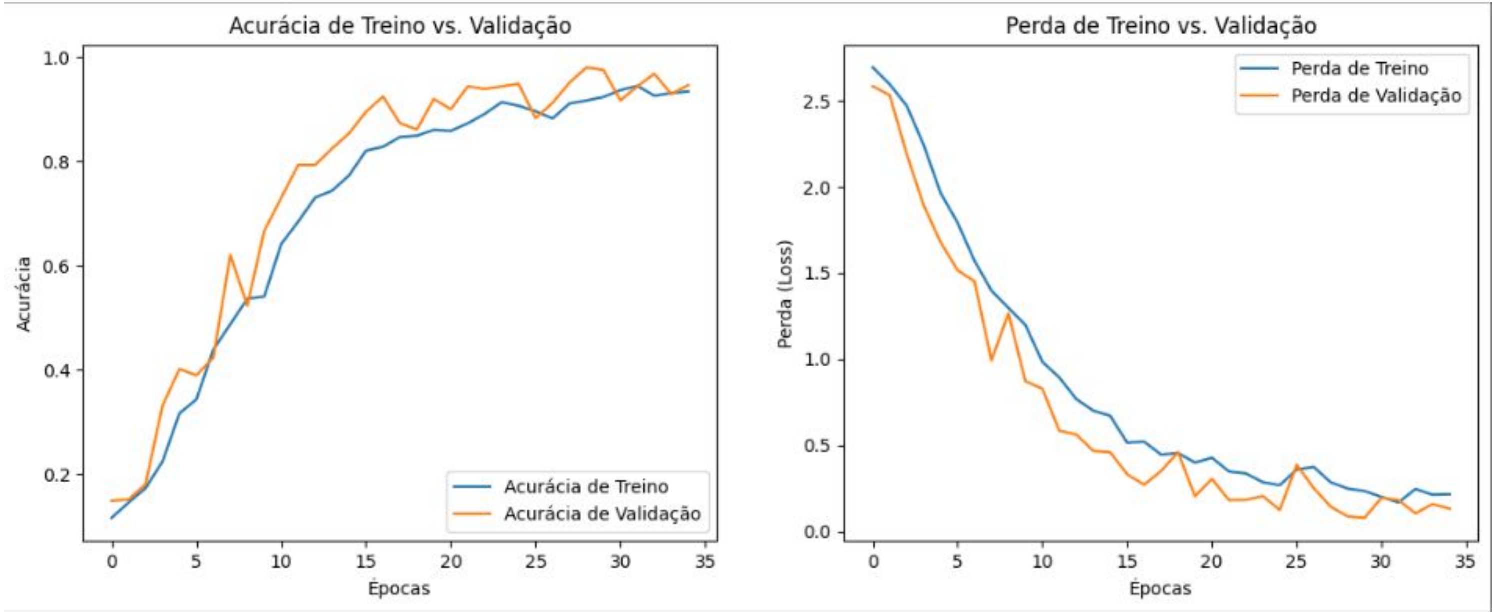
Otimizador ADAM - Adaptive moment estimation

RMSprop

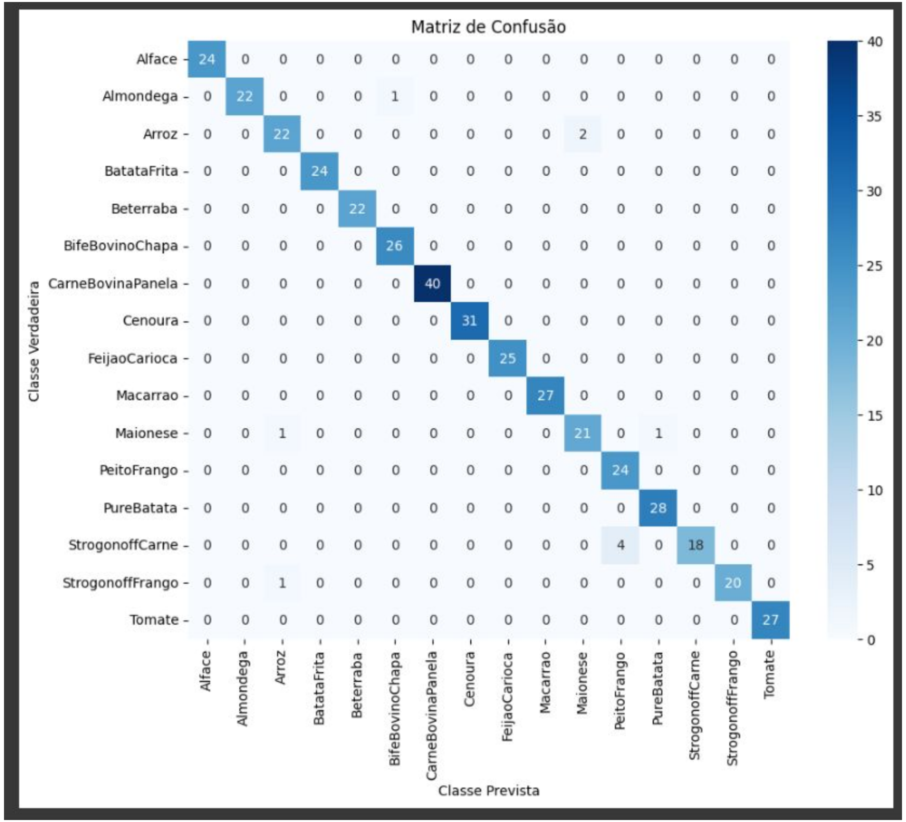
Treinamento do modelo

```
EPOCHS = 100 # EarlyStopping vai cuidar do resto

history_otimizado = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    callbacks=[model_checkpoint, early_stopping, reduce_lr] # Adiciona a lista de callbacks
)
```

Resultados - modelo otimizado



Resultados - modelo otimizado

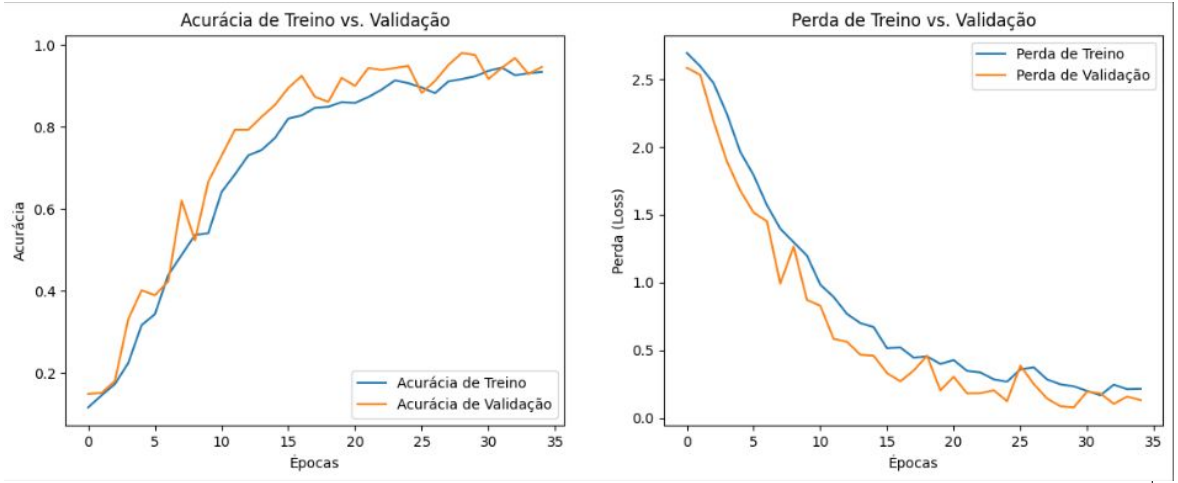


Resultados - modelo otimizado

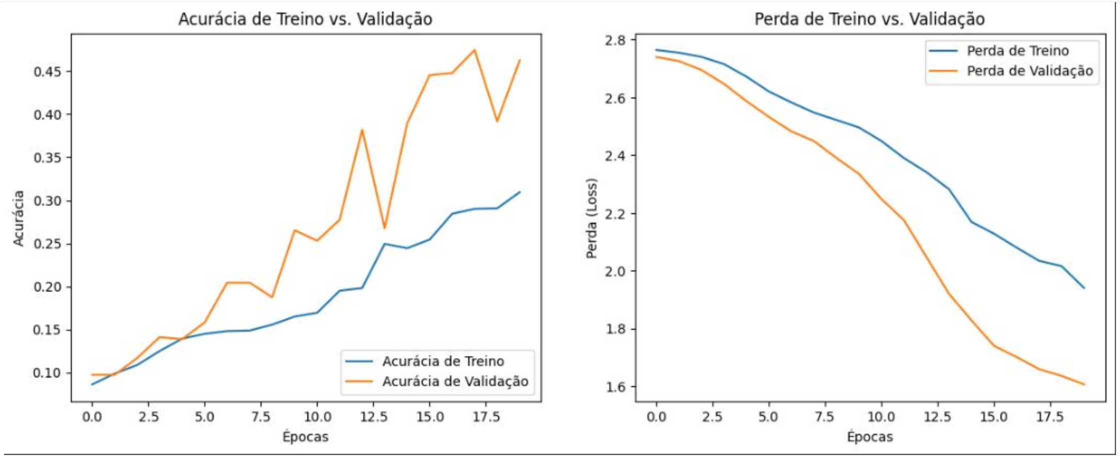
	precision	recall	f1-score	support
Alface	1.00	1.00	1.00	24
Almondega	1.00	0.96	0.98	23
Arroz	0.92	0.92	0.92	24
BatataFrita	1.00	1.00	1.00	24
Beterraba	1.00	1.00	1.00	22
BifeBovinoChapa	0.96	1.00	0.98	26
CarneBovinaPanela	1.00	1.00	1.00	40
Cenoura	1.00	1.00	1.00	31
FeijaoCarioca	1.00	1.00	1.00	25
Macarrao	1.00	1.00	1.00	27
Maionese	0.91	0.91	0.91	23
PeitoFrango	0.86	1.00	0.92	24
PureBatata	0.97	1.00	0.98	28
StrogonoffCarne	1.00	0.82	0.90	22
StrogonoffFrango	1.00	0.95	0.98	21
Tomate	1.00	1.00	1.00	27
accuracy			0.98	411
macro avg	0.98	0.97	0.97	411
weighted avg	0.98	0.98	0.98	411

Comparação

otimizado



base



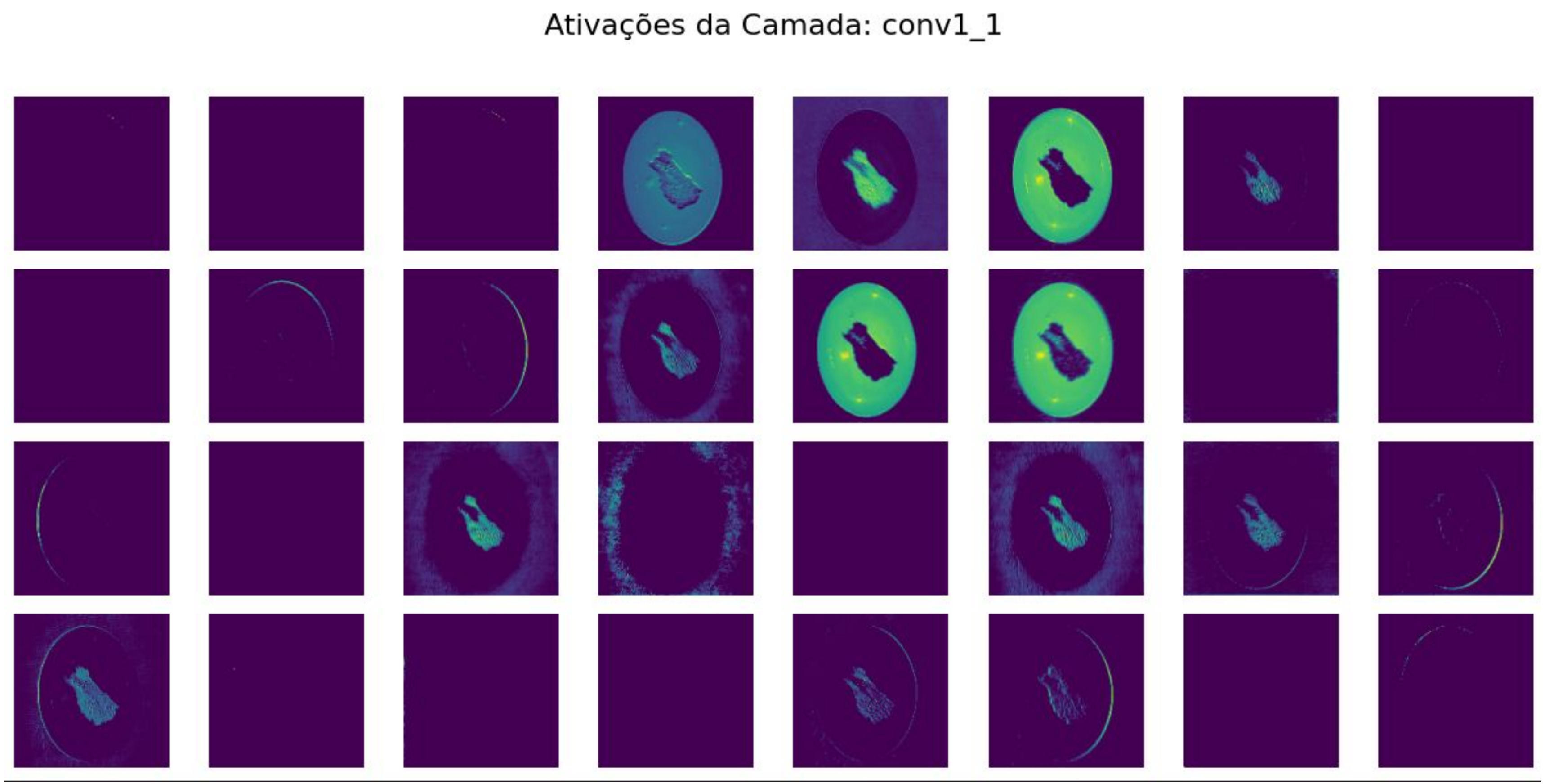
Comparação - Acurácia

Modelo base -> 50%

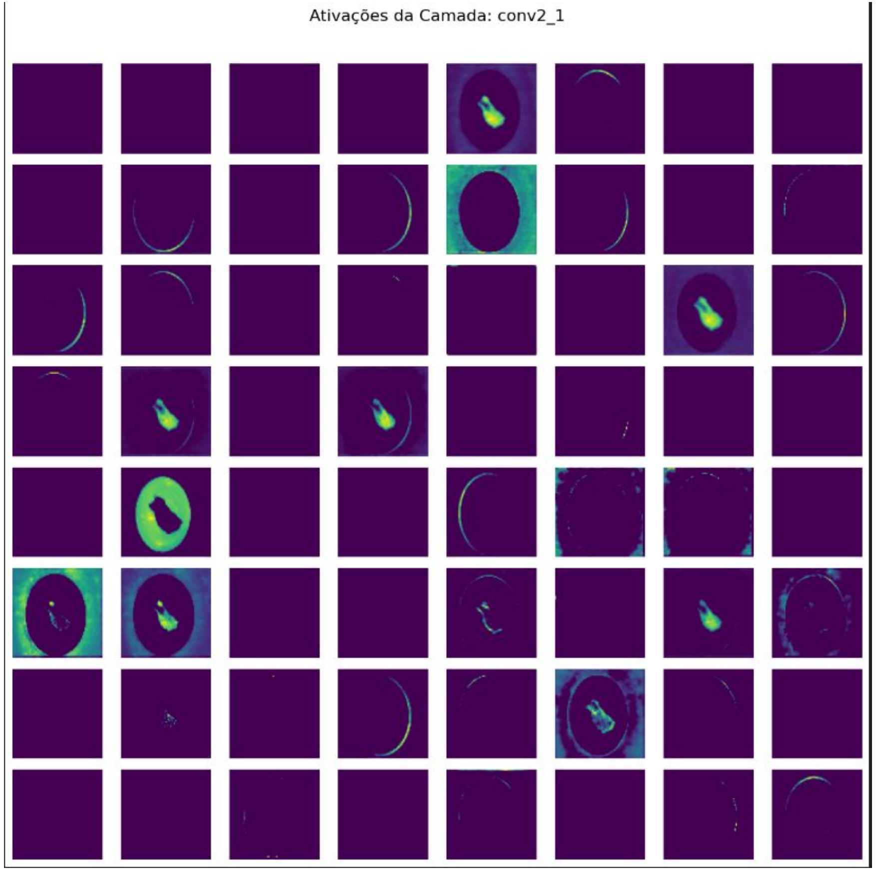
Modelo Otimizado -> 98%

Análise de erros na prática

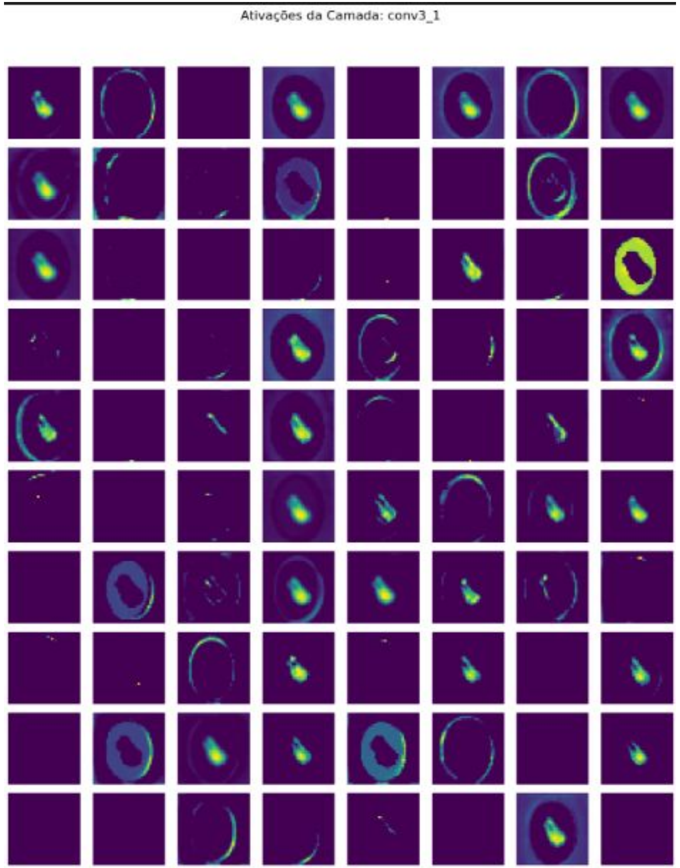
Visualizando as camadas de ativação

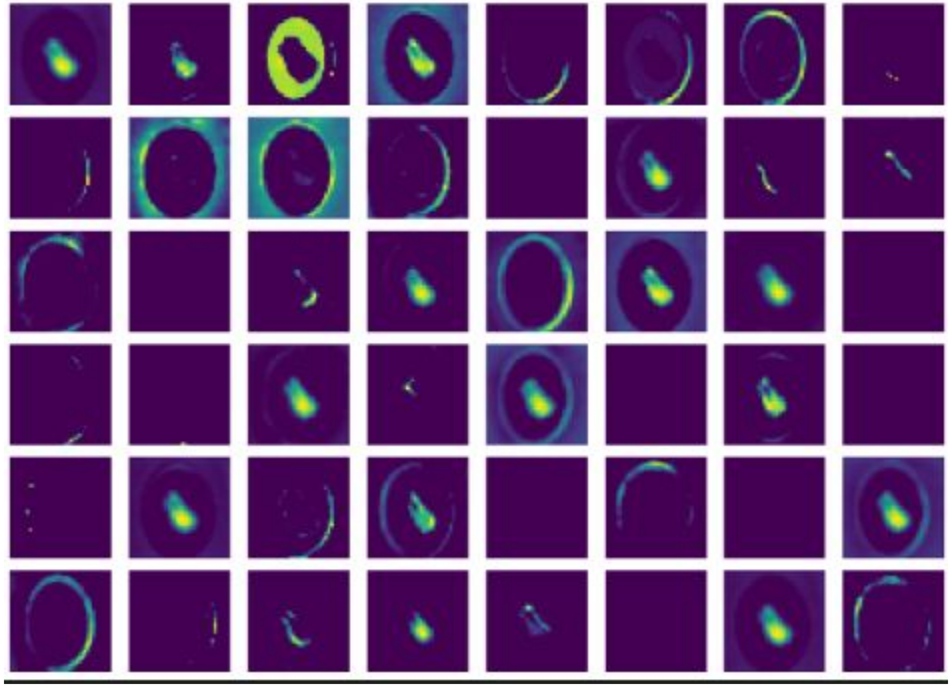


Visualizando as camadas de ativação

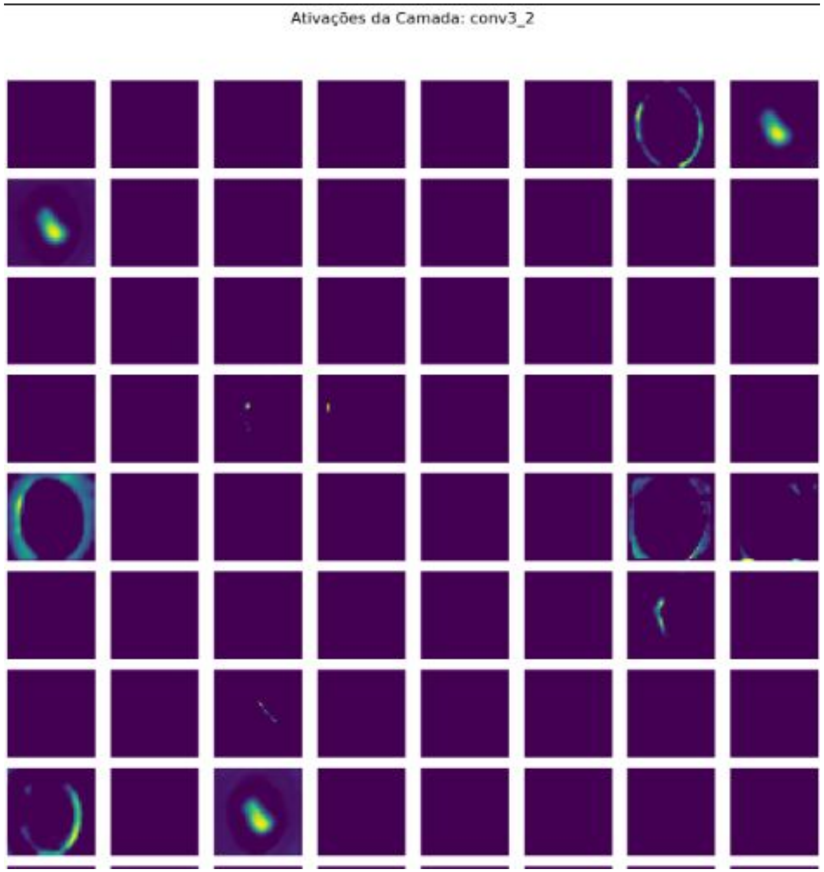


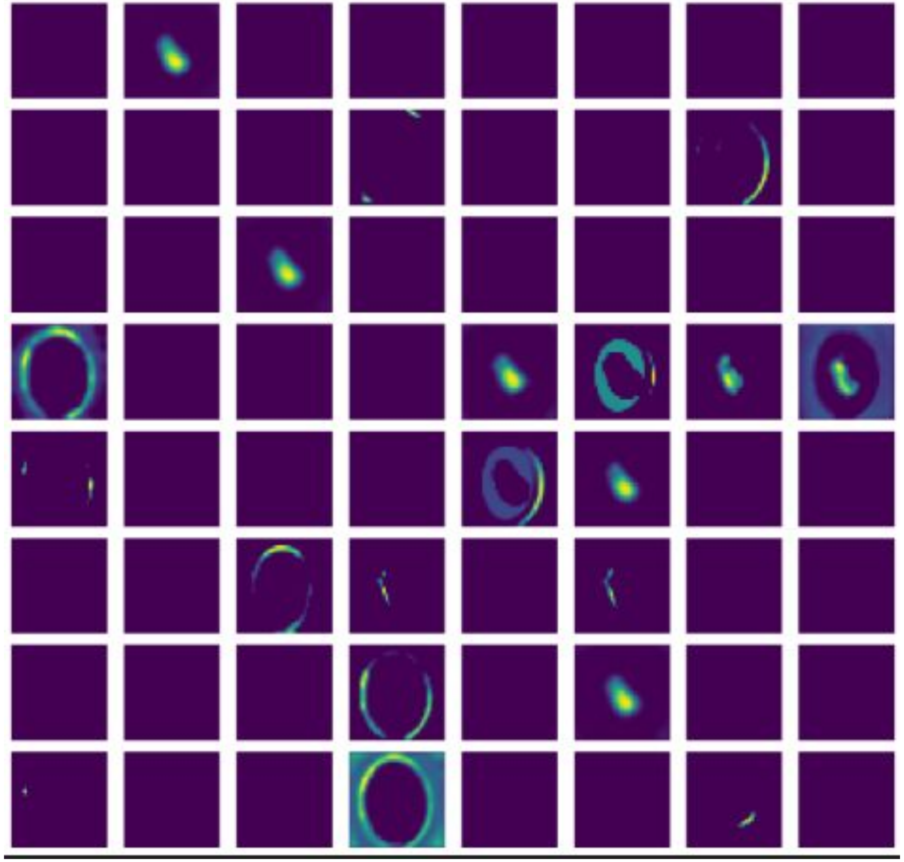
Visualizando as camadas de ativação





Visualizando as camadas de ativação





Próximos passos

Demonstração