# treino

Para adaptar o processo de machine learning para reconhecer imagens de maçãs, limões e cebolas, precisaremos usar um modelo de deep learning adequado para classificação de imagens. A biblioteca Keras, que é uma API de alto nível para redes neurais, é uma boa escolha para essa tarefa. Vou apresentar um exemplo usando uma rede neural convolucional (CNN), que é adequada para reconhecimento de imagens.

1. \*\*Organizar as Imagens\*\*:

Certifique-se de que as imagens estão organizadas nas seguintes diretórias:

```

- dataset/

- apples/

- apple1.png

- apple2.png

- ...

- lemons/

- lemon1.png

- lemon2.png

- ...

- onions/

- onion1.png

- onion2.png

- ...

```

2. \*\*Instalar as Bibliotecas Necessárias\*\*:

Certifique-se de que tem instaladas as bibliotecas necessárias: TensorFlow, Keras, NumPy, Matplotlib, entre outras.

```bash

pip install tensorflow keras numpy matplotlib

```

3. \*\*Código para Treinar o Modelo\*\*:

```python

import os

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

# Definir o caminho das diretórias

base\_dir = 'dataset'

train\_dir = os.path.join(base\_dir)

# Parâmetros

img\_width, img\_height = 150, 150

batch\_size = 32

epochs = 15

# Data augmentation e normalização

train\_datagen = ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

validation\_split=0.2)

# Geradores de dados

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical',

subset='training')

validation\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical',

subset='validation')

# Construir o modelo

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_width, img\_height, 3)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(512, activation='relu'),

Dropout(0.5),

Dense(3, activation='softmax') # 3 classes: apples, lemons, onions

])

# Compilar o modelo

model.compile(optimizer=Adam(),

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Treinar o modelo

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // batch\_size,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // batch\_size,

epochs=epochs)

# Avaliar o modelo

plt.plot(history.history['accuracy'], label='Acurácia de Treino')

plt.plot(history.history['val\_accuracy'], label='Acurácia de Validação')

plt.xlabel('Época')

plt.ylabel('Acurácia')

plt.legend()

plt.show()

```

\*\*Explicação do Código\*\*:

- \*\*ImageDataGenerator\*\*: Utilizado para realizar data augmentation e normalização das imagens.

- \*\*Sequential Model\*\*: Criamos uma CNN com várias camadas convolucionais e de pooling, seguida de camadas densas para classificação.

- \*\*Compilação e Treinamento\*\*: O modelo é compilado com o otimizador Adam e treinado usando os dados gerados pelos `ImageDataGenerator`.

Esse código básico pode ser ajustado e melhorado conforme a quantidade e complexidade dos dados aumentarem.

Deployment

Found 784 images belonging to 5 classes.

Found 194 images belonging to 5 classes.

c:\lang\Python312\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

Epoch 1/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m19s[0m 643ms/step - accuracy: 0.5069 - loss: 1.4143 - val\_accuracy: 0.9688 - val\_loss: 0.1082

Epoch 2/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 2ms/step - accuracy: 1.0000 - loss: 0.0618 - val\_accuracy: 1.0000 - val\_loss: 0.1045

Epoch 3/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m16s[0m 602ms/step - accuracy: 0.9401 - loss: 0.2290 - val\_accuracy: 0.9479 - val\_loss: 0.1612

Epoch 4/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 0.8750 - loss: 0.2859 - val\_accuracy: 1.0000 - val\_loss: 0.0015

Epoch 5/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 610ms/step - accuracy: 0.9463 - loss: 0.1581 - val\_accuracy: 0.9583 - val\_loss: 0.1462

Epoch 6/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 0.8750 - loss: 0.2707 - val\_accuracy: 1.0000 - val\_loss: 0.0659

Epoch 7/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 638ms/step - accuracy: 0.9689 - loss: 0.1013 - val\_accuracy: 0.9531 - val\_loss: 0.1423

Epoch 8/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 0.9688 - loss: 0.1907 - val\_accuracy: 1.0000 - val\_loss: 0.0079

Epoch 9/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 645ms/step - accuracy: 0.9626 - loss: 0.1255 - val\_accuracy: 0.9844 - val\_loss: 0.0638

Epoch 10/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 1.0000 - loss: 0.0700 - val\_accuracy: 1.0000 - val\_loss: 0.0235

Epoch 11/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 629ms/step - accuracy: 0.9756 - loss: 0.1054 - val\_accuracy: 0.9792 - val\_loss: 0.0567

Epoch 12/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 0.9688 - loss: 0.0587 - val\_accuracy: 1.0000 - val\_loss: 0.0012

Epoch 13/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 630ms/step - accuracy: 0.9720 - loss: 0.0628 - val\_accuracy: 0.9792 - val\_loss: 0.0338

Epoch 14/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 1.0000 - loss: 0.0170 - val\_accuracy: 1.0000 - val\_loss: 5.1615e-05

Epoch 15/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m17s[0m 606ms/step - accuracy: 0.9866 - loss: 0.0381 - val\_accuracy: 0.9896 - val\_loss: 0.0367

Epoch 16/16

[1m24/24[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 1.0000 - loss: 0.0082 - val\_accuracy: 1.0000 - val\_loss: 0.0000e+00

<Figure size 640x480 with 1 Axes>

A graph of a graph with blue and orange lines

Description automatically generated