Inteligência Artificial

Lista Extra - Deep Learning

Aluno: Otávio Augusto de Assis Ferreira Monteiro

Belo Horizonte, 2025

Link do código:
https://github.com/otavioaugustoafm/Faculdade/blob/main/IA/Listas/Lista%20Extra/dogs_vs_cats.ipynb

Questão 1.1)

```python
import os
from PIL import Image

def limpar_imagens_corrompidas(folder_path):
    print(f"--- Verificando imagens em: {folder_path} ---")
    deleted_files = 0

    # Percorre todas as pastas e subpastas
    for root, dirs, files in os.walk(folder_path):
        for filename in files:
            file_path = os.path.join(root, filename)

            try:
                # Tenta abrir a imagem
                with Image.open(file_path) as img:
                    img.verify() # Verifica a integridade do arquivo
            except (IOError, SyntaxError, Image.UnidentifiedImageError) as e:
                # Se der erro, deleta o arquivo
                print(f"Arquivo corrompido removido: {filename}")
                os.remove(file_path)
                deleted_files += 1

    if deleted_files == 0:
        print("Nenhuma imagem corrompida encontrada.")
    else:
        print(f"Total de arquivos removidos: {deleted_files}")

# Chama a função na sua pasta base
limpar_imagens_corrompidas(base_dir)
```

```
✓ 2m 25.4s

--- Verificando imagens em: ./dogs_vs_cats ---
Arquivo corrompido removido: 666.jpg
Arquivo corrompido removido: Thumbs.db
Arquivo corrompido removido: 11702.jpg
C:\Users\otavi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p
  warnings.warn(str(msg))
Arquivo corrompido removido: Thumbs.db
Total de arquivos removidos: 4
```

A imagem acima compreende a exclusão de imagens corrompidas ou que estavam dando algum tipo de erro durante os processos de aprendizado. Como podemos notar, 4 itens foram removidos.

```python
print(f"Lendo imagens de: {os.path.abspath(base_dir)}")

# --- Configuração dos Geradores com Separação Automática ---

# Definimos o Data Augmentation E a divisão de validação (20%) aqui
train_datagen = ImageDataGenerator(
    rescale=1./255,             # Normalização
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2        # Separa 20% para teste automaticamente
)

# Gerador de Treino (usa 80% dos dados)
train_generator = train_datagen.flow_from_directory(
    base_dir,                   # Aponta para a pasta dogs_vs_cats
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training'           # <--- Pega apenas a parte de treino
)

# Gerador de Teste/Validação (usa 20% dos dados)
test_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=False,              # Importante manter False para a Matriz de Confusão
    subset='validation'         # <--- Pega apenas a parte de validação
)
```

Aqui é onde foram separados os conjuntos de treino e de teste. Escolhi definir que o conjunto de teste utilizaria 20% das imagens totais e o de treino 80%. Também foi feita a padronização de tamanho, levando todas as imagens para 150x150.

Questão 1.2)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

Total params: 3,453,121 (13.17 MB)

Trainable params: 3,453,121 (13.17 MB)

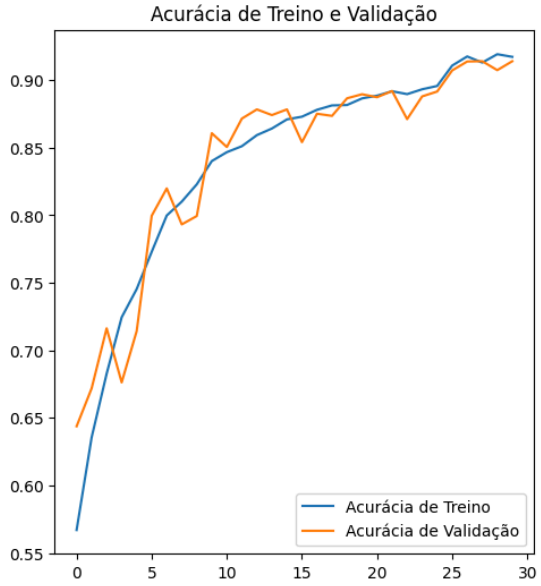
Non-trainable params: 0 (0.00 B)

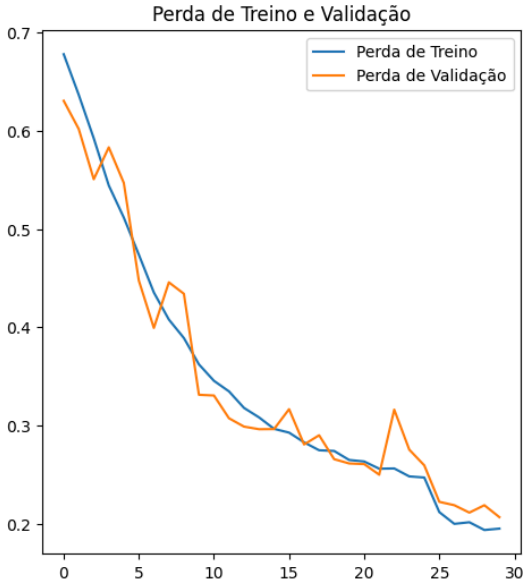
Essa imagem representa a construção e compilação da CNN. Os 3 blocos solicitados foram criados na célula que gerou essa tabela (Conv2D + MaxPooling2D). O Flatten + Dense é explorado nela também, assim como a camada final de saída com ativação sigmoid. O hiperparâmetro de learning_rate foi experimentado e utilizado por aqui, onde escolhi usar 0,001 para um aprendizado bem preciso.

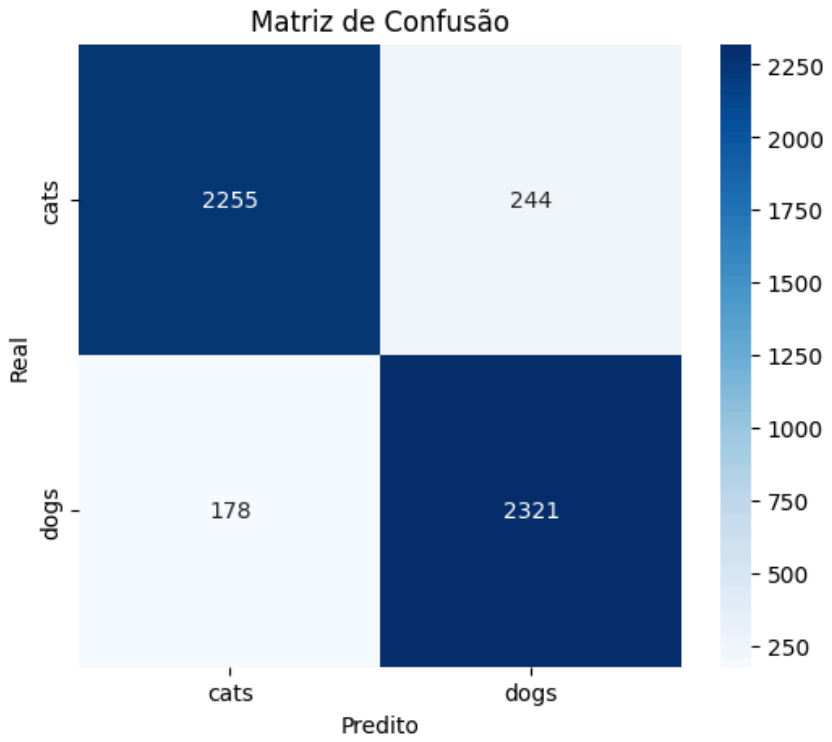
```
Iniciando treinamento...
Epoch 1/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 254ms/step - accuracy: 0.5670 - loss: 0.6779 - val_accuracy: 0.6438 - val_loss: 0.6305 - learning_rate: 0.0010
Epoch 2/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 158s 253ms/step - accuracy: 0.6356 - loss: 0.6366 - val_accuracy: 0.6719 - val_loss: 0.6018 - learning_rate: 0.0010
Epoch 3/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 256ms/step - accuracy: 0.6827 - loss: 0.5923 - val_accuracy: 0.7163 - val_loss: 0.5507 - learning_rate: 0.0010
Epoch 4/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 256ms/step - accuracy: 0.7245 - loss: 0.5444 - val_accuracy: 0.6763 - val_loss: 0.5832 - learning_rate: 0.0010
Epoch 5/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 257ms/step - accuracy: 0.7455 - loss: 0.5117 - val_accuracy: 0.7143 - val_loss: 0.5470 - learning_rate: 0.0010
Epoch 6/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 256ms/step - accuracy: 0.7730 - loss: 0.4740 - val_accuracy: 0.7997 - val_loss: 0.4477 - learning_rate: 0.0010
Epoch 7/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 256ms/step - accuracy: 0.7998 - loss: 0.4355 - val_accuracy: 0.8199 - val_loss: 0.3994 - learning_rate: 0.0010
Epoch 8/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 160s 256ms/step - accuracy: 0.8102 - loss: 0.4079 - val_accuracy: 0.7933 - val_loss: 0.4458 - learning_rate: 0.0010
Epoch 9/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 158s 253ms/step - accuracy: 0.8230 - loss: 0.3891 - val_accuracy: 0.7995 - val_loss: 0.4342 - learning_rate: 0.0010
Epoch 10/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 159s 254ms/step - accuracy: 0.8401 - loss: 0.3624 - val_accuracy: 0.8608 - val_loss: 0.3314 - learning_rate: 0.0010
Epoch 11/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 158s 252ms/step - accuracy: 0.8468 - loss: 0.3456 - val_accuracy: 0.8506 - val_loss: 0.3307 - learning_rate: 0.0010
Epoch 12/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 158s 253ms/step - accuracy: 0.8511 - loss: 0.3349 - val_accuracy: 0.8716 - val_loss: 0.3075 - learning_rate: 0.0010
...
Epoch 30/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 158s 253ms/step - accuracy: 0.9173 - loss: 0.1952 - val_accuracy: 0.9141 - val_loss: 0.2069 - learning_rate: 2.0000e-04
Restoring model weights from the end of the best epoch: 30.
Treinamento concluído.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Aqui temos o treinamento em si que ocorreu durante 30 épocas e durou aproximadamente 80 minutos.

Plotagem dos gráficos:








```
Relatório de Classificação:

              precision    recall  f1-score   support

        cats       0.93      0.90      0.91      2499
        dogs       0.90      0.93      0.92      2499

    accuracy                           0.92      4998
   macro avg       0.92      0.92      0.92      4998
weighted avg       0.92      0.92      0.92      4998
```

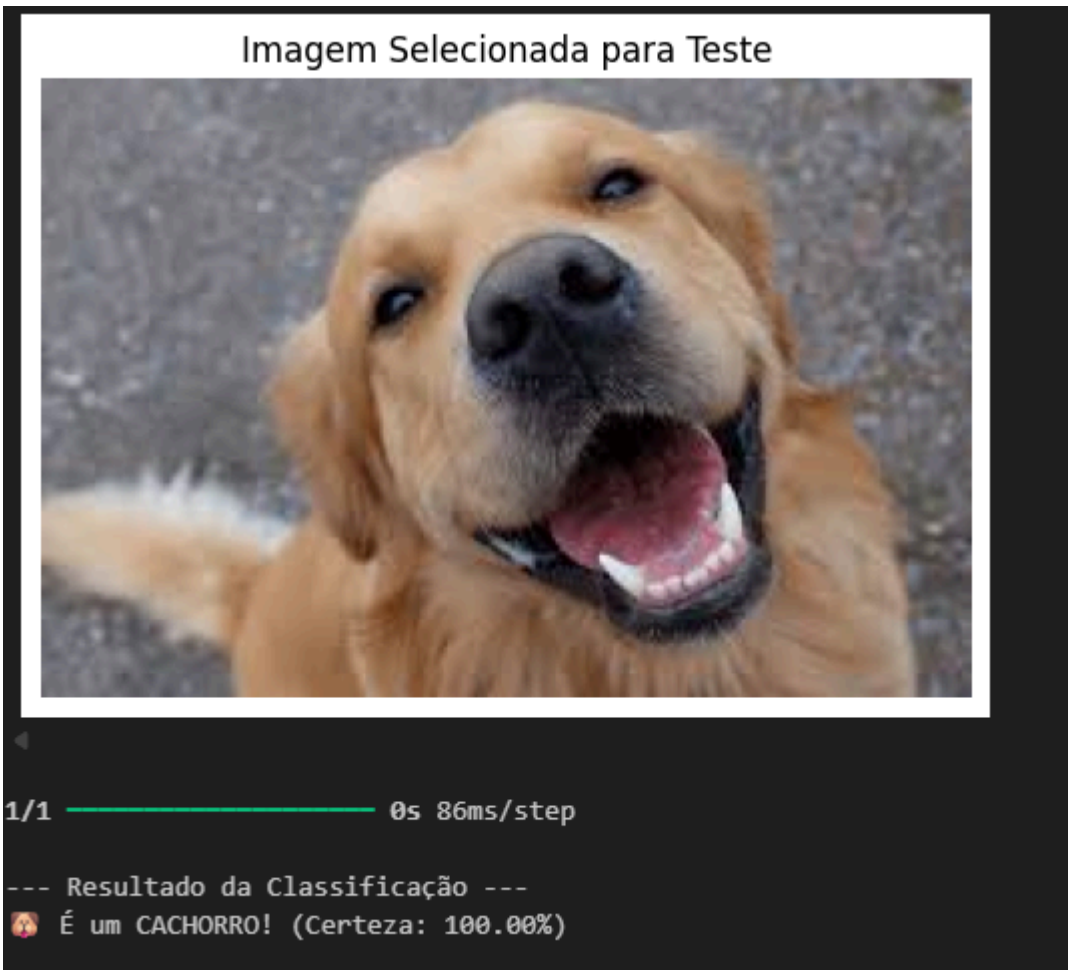
Conclusão:

O treinamento da CNN trouxe um resultado satisfatório e estável durante as 30 épocas de duração. As curvas de aprendizado demonstraram uma boa consistência, com a acurácia e validação crescendo simultaneamente.

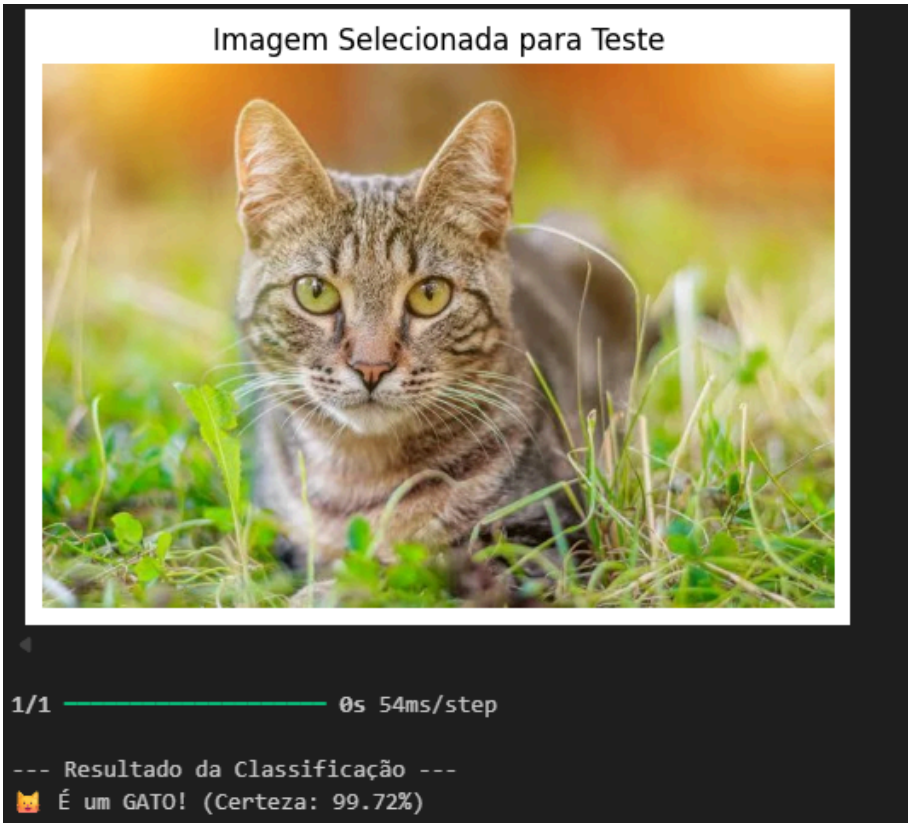
Tratando das métricas quantitativas agora, o modelo atingiu 92% de acurácia no conjunto de teste. O F1-Score de 0.91 para gatos e 0.92 para cachorros aponta para um desempenho equilibrado entre as duas classes. A matriz de confusão, por sua vez, revela uma maior facilidade para identificar cachorros, com um Recall de 0.93, do que identificar gatos, com Recall de 0.9.

No geral, de um total de cerca de 5000 imagens de teste, o modelo confundiu gatos com cachorros 244 vezes, enquanto o erro inverso ocorreu apenas 178 vezes.

Testes com imagens da internet



Identificado com sucesso!



Identificado com sucesso!

É um rato, mas a CNN entendeu como se fosse um gato.