Lista #12

Curso: Ciência da Computação

Disciplina: Inteligência Artificial

Profª. Cristiane Neri Nobre

Data de entrega: 08/06

Valor: 5 pontos

Aluno: Lucas Henrique Rocha Hauck

Github: https://github.com/o-hauck/IA

1. Preparação dos dados

- Faça o download e carregamento da base de imagens.
- Separe os dados em treino, validação e teste (ex: 70% treino, 15% validação, 15% teste).
- Aplique técnicas de pré-processamento de imagens:
    - Redimensionamento para tamanho padrão (ex: 150x150 ou 224x224) ○ Normalização dos pixels
    - Aumento de dados: rotação, inversão horizontal, zoom etc.

Nessa etapa utilizei a API do kaggle para baixar a base de dados diretamente no google collab:

```
[2] from google.colab import files
    files.upload()
```

```
Procurar... kaggle.json
kaggle.json(application/json) - 66 bytes, last modified: n/a - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"lucashauck","key":[redacted]'}'}
```

```
# Cria a pasta .kaggle e move o arquivo json para lá
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# Define as permissões corretas para o arquivo
!chmod 600 ~/.kaggle/kaggle.json

# Baixa a base de dados "dogs-vs-cats" do Kaggle
# A flag -c indica o nome da competição
!kaggle competitions download -c dogs-vs-cats

# Descompacta o arquivo train.zip de forma silenciosa (-q)
# Isso criará uma pasta chamada 'train'
!unzip -q dogs-vs-cats.zip -d .

# Descompacta o arquivo train.zip de forma silenciosa (-q)
# Isso criará uma pasta chamada 'train'
!unzip -q train.zip -d .
```

```
Downloading dogs-vs-cats.zip to /content
 98% 792M/812M [00:05<00:00, 120MB/s]
100% 812M/812M [00:05<00:00, 166MB/s]
```

Depois preparei os dados:

1. Separar as imagens em pastas diferentes:

```python
import os
import shutil

# 1. Definir os caminhos no ambiente do Google Colab
# A pasta original com as imagens descompactadas
original_dir = '/content/train'

# A nova pasta onde vamos organizar os dados
base_dir = '/content/data_gatos_vs_caes'

# Não execute se a pasta base já existir (para evitar erros se rodar a célula de novo)
if not os.path.exists(base_dir):
    os.mkdir(base_dir)

    # 2. Criar os diretórios para gatos e cachorros dentro da pasta base
    cats_dir = os.path.join(base_dir, 'cats')
    os.mkdir(cats_dir)

    dogs_dir = os.path.join(base_dir, 'dogs')
    os.mkdir(dogs_dir)

    # 3. Pegar a lista de todos os nomes de arquivo da pasta original
    fnames = os.listdir(original_dir)

    print(f"Iniciando a movimentação de {len(fnames)} arquivos...")

    # 4. Mover os arquivos para as pastas corretas
    for fname in fnames:
        # Verifica se o nome do arquivo começa com 'cat'
        if fname.startswith('cat'):
            # Cria o caminho de origem e destino
            src = os.path.join(original_dir, fname)
            dst = os.path.join(cats_dir, fname)
            # Move o arquivo
            shutil.move(src, dst)

        # Verifica se o nome do arquivo começa com 'dog'
        elif fname.startswith('dog'):
            src = os.path.join(original_dir, fname)
            dst = os.path.join(dogs_dir, fname)
            shutil.move(src, dst)

    # 5. Imprimir um resumo
    print("\nOrganização dos arquivos concluída!")
    print(f"Total de imagens de gatos: {len(os.listdir(cats_dir))}")
    print(f"Total de imagens de cachorros: {len(os.listdir(dogs_dir))}")

else:
    print(f"A pasta '{base_dir}' já existe. Os arquivos provavelmente já foram organizados.")
```

2. Divisão entre treino, validação e teste. Pre processamento:
    a. Nessa etapa devidi os dados entre treino, validação e teste (70/15/15) e aumentei os dados retacionando, invertendo e dando zoom.

```python
# Célula 4: Carregar os dados e aplicar pré-processamento
import tensorflow as tf

# Agora você pode acessar o Keras com tf.keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential


# Parâmetros
img_height = 150
img_width = 150
batch_size = 32
data_dir = '/content/data_gatos_vs_caes' # Caminho da pasta organizada

# Carregar dados de treino (70%)
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.3, # Reservar 30% para validação e teste
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Carregar os 30% restantes
val_test_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.3,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Dividir os 30% em 15% para validação e 15% para teste
val_test_batches = tf.data.experimental.cardinality(val_test_ds)
val_ds = val_test_ds.take(val_test_batches // 2)
test_ds = val_test_ds.skip(val_test_batches // 2)

class_names = train_ds.class_names
print("Classes:", class_names)

# Otimizar performance do pipeline de dados
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Camada para aumento de dados
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal"),
  layers.RandomRotation(0.1),
  layers.RandomZoom(0.1),
])

# Camada de normalização de pixels (0-255 para 0-1)
normalization_layer = layers.Rescaling(1./255)
```

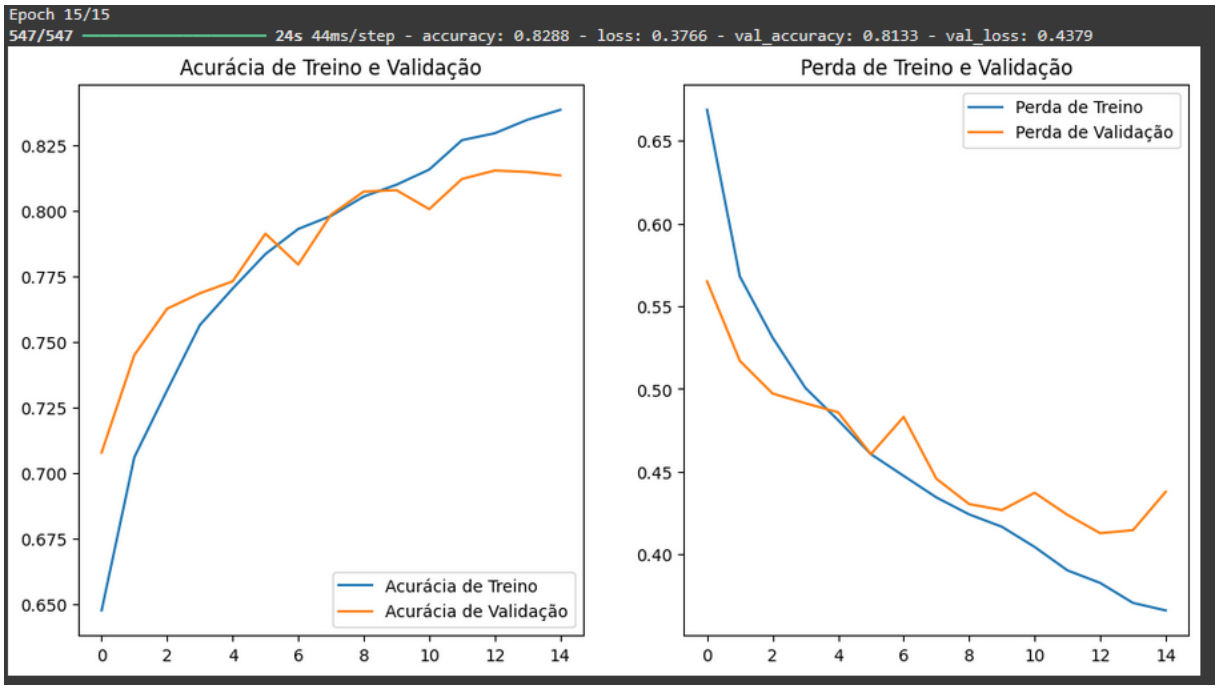
2. Construção e treinamento de uma CNN

Construção:

```python
# Célula para construir o modelo
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

model = Sequential([
    # Adiciona as camadas de pré-processamento no inicio do modelo
    layers.Input(shape=(img_height, img_width, 3)),
    normalization_layer,
    data_augmentation,

    # Bloco 1: Conv2D + MaxPooling2D
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Bloco 2: Conv2D + MaxPooling2D
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Camada Flatten para transformar os dados em um vetor
    layers.Flatten(),

    # Camada Densa (totalmente conectada)
    layers.Dense(512, activation='relu'),

    # Camada final de saída com ativação sigmoid para classificação binária
    layers.Dense(1, activation='sigmoid')
])

# Compila o modelo
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Treino:

```python
# Célula para treinar o modelo
epochs = 15 # O exercício pede pelo menos 10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)

# Célula para plotar os gráficos
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Acurácia de Treino')
plt.plot(epochs_range, val_acc, label='Acurácia de Validação')
plt.legend(loc='lower right')
plt.title('Acurácia de Treino e Validação')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Perda de Treino')
plt.plot(epochs_range, val_loss, label='Perda de Validação')
plt.legend(loc='upper right')
plt.title('Perda de Treino e Validação')
plt.show()
```



Treinamento feito em 15 épocas, antes notei que estava demorando demais então mudei o ambiente para utilizar GPUs e o treinamento foi feito de forma muito mais rápida.

Observando os gráficos do treino e validação podemos observar que a partir da época 8 a perda de validação para de cair significativamente e fica variando enquanto a perda de treino continua caindo, isso é um sinal claro de overfitting do modelo. Na próxima vez tentarei utilizar 8 épocas para evitar isso ou utilizar uma técnica de dropout para evitar que os neurônios “decorem” os dados de treino.

3. Avaliação e testes

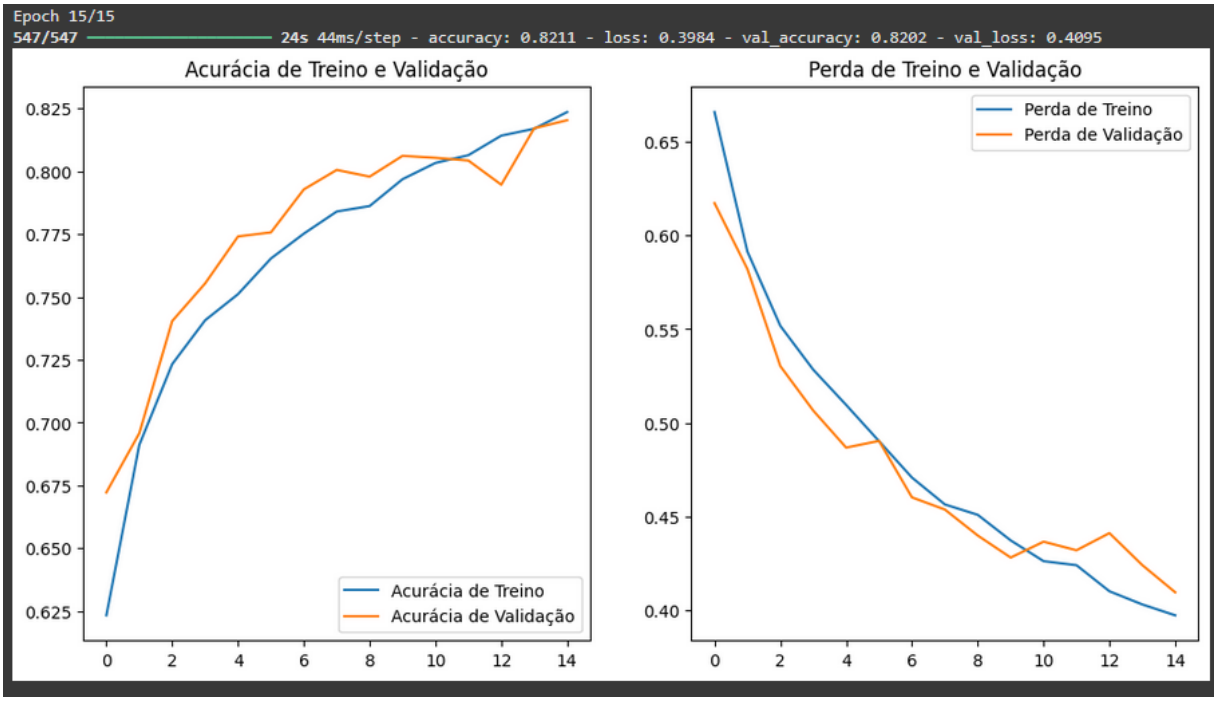
```
Acurácia no conjunto de teste: 0.81
118/118 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step

Relatório de Classificação:
              precision    recall  f1-score   support

        cats       0.82      0.80      0.81      1858
        dogs       0.81      0.83      0.82      1898

    accuracy                           0.81      3756
   macro avg       0.81      0.81      0.81      3756
weighted avg       0.81      0.81      0.81      3756
```

De acordo com as métricas obtidas podemos perceber que temos uma acurácia de 81% e um ótimo balanceamento entre precisão e recall para ambas as classes, com um f1 score de 81%

para gatos e 82% para cachorros. A utilização da técnica de dropout poderia levar a resultados melhores.

Mudanças ao adicionar a camada de dropout:

layers.Dropout(0.5)



Podemos observar que com o dropout a validação fica bem mais próxima do treino, isso normalmente significa que existe menos overfitting.

```
Avaliando o modelo no conjunto de teste...
118/118 ━━━━━━━━━━━━━━━━━━━━ 8s 28ms/step - accuracy: 0.8337 - loss: 0.3786

Acurácia no conjunto de teste: 0.82
118/118 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step

Relatório de Classificação:
              precision    recall  f1-score   support

        cats       0.84      0.79      0.81      1858
        dogs       0.81      0.86      0.83      1898

    accuracy                           0.82      3756
   macro avg       0.82      0.82      0.82      3756
weighted avg       0.82      0.82      0.82      3756
```

Observando o resultado da avaliação no conjunto de teste podemos observar uma melhora de 1% na acurácia, 2% na precisão para gatos, piora de 1% no recall para gatos, melhora de

3% no recall para cachorros. O f1-score para gatos ficou o mesmo, mas para cachorros aumento 1%. Por isso percebemos que o modelo ficou melhor num geral.

Teste feito com fotos das cachorras da minha namorada:

Teste feito com imagem de fato encontrada no google: