Lista 12

https://colab.research.google.com/drive/1hAs_onnCv-dOveHn8L7O2oGhD_58e-cR?usp=sharing

🐶🐱 Relatório Técnico — Classificação de Cães e Gatos com CNN

🎯 Objetivo

Desenvolver um modelo de **Rede Neural Convolucional (CNN)** capaz de classificar imagens entre **Cães (1)** e **Gatos (0)**, utilizando o dataset de aproximadamente **25.000 imagens**.

📂 Etapas do Projeto

1️⃣ Importação e Organização dos Dados

🔗 Download e Descompactação

- O dataset é obtido no link:

```
https://download.microsoft.com/download/3/e/1/3e1c3f21-ecdb-4869-8368-6deba77b919f/kagglecatsanddogs_5340.zip
```

- Estrutura após descompactação:

```
PetImages/
├── Cat/
└── Dog/
```

📑 Criação do DataFrame

O código percorre as pastas `Cat` e `Dog` e associa os rótulos:

- **0 → Cat**
- **1 → Dog**

```
input_path = []
label = []

for class_name in os.listdir("PetImages"):
    for path in os.listdir("PetImages/"+class_name):
        if class_name == 'Cat':
            label.append(0)
        else:
            label.append(1)
        input_path.append(os.path.join("PetImages", class_name, path))
print(input_path[0], label[0])
```

🧹 Limpeza de Dados

- Remoção de arquivos inválidos, como `Thumbs.db` e imagens corrompidas.

```
# delete db files
df = df[df['images']!='PetImages/Dog/Thumbs.db']
df = df[df['images']!='PetImages/Cat/Thumbs.db']
df = df[df['images']!='PetImages/Cat/666.jpg']
df = df[df['images']!='PetImages/Dog/11702.jpg']
len(df)
```

2️⃣ Análise Exploratória de Dados (EDA)

- **Visualização de Amostras**:

- Grades de imagens para inspeção visual, tanto de cães como de gatos.

```
# to display grid of images
plt.figure(figsize=(25,25))
temp = df[df['label']==1]['images']
start = random.randint(0, len(temp))
files = temp[start:start+25]

for index, file in enumerate(files):
    plt.subplot(5,5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title('Dogs')
    plt.axis('off')
```

- **Distribuição das Classes**:

```python
# to display grid of images
plt.figure(figsize=(25,25))
temp = df[df['label']==0]['images']
start = random.randint(0, len(temp))
files = temp[start:start+25]

for index, file in enumerate(files):
    plt.subplot(5,5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title('Cats')
    plt.axis('off')
```

✔️ Classes estão relativamente balanceadas.

3️⃣ Pré-processamento das Imagens

✂️ Divisão em Treino e Teste

- 70% para treino, 15% para validação e 15% teste:

```python
from sklearn.model_selection import train_test_split

# Dividir em treino (70%) + temp (30%)
train, temp = train_test_split(df, test_size=0.30, random_state=42)

# Dividir temp em validação (15%) e teste (15%)
val, test = train_test_split(temp, test_size=0.5, random_state=42)

print(f'Treinamento: {len(train)} imagens')
print(f'Validação: {len(val)} imagens')
print(f'Teste: {len(test)} imagens')
```

```
Treinamento: 17498 imagens
Validação: 3750 imagens
Teste: 3750 imagens
```

🔄 Geradores de Dados

- Utiliza **ImageDataGenerator** para:
    - **Normalizar imagens** (escala dos pixels para [0,1]).
    - **Aumentar dados** no treino com:
        - Rotações
        - Zoom
        - Shear
        - Flip horizontal

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(
    rescale = 1./255,  # normalization of images
    rotation_range = 40, # augmention of images to avoid overfitting
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest'
)

val_generator = ImageDataGenerator(rescale = 1./255)
```

🔢 Criação dos Iteradores

```
train_iterator = train_generator.flow_from_dataframe(
    train,
    x_col='images',
    y_col='label',
    target_size=(128,128),
    batch_size=512,
    class_mode='binary'
)

val_iterator = val_generator.flow_from_dataframe(
    test,
    x_col='images',
    y_col='label',
    target_size=(128,128),
    batch_size=512,
    class_mode='binary'
)
```

4️⃣ Construção da CNN

🏗️ Arquitetura do Modelo

```
from keras import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense

model = Sequential([
                    Conv2D(16, (3,3), activation='relu', input_shape=(128,128,3)),
                    MaxPool2D((2,2)),
                    Conv2D(32, (3,3), activation='relu'),
                    MaxPool2D((2,2)),
                    Conv2D(64, (3,3), activation='relu'),
                    MaxPool2D((2,2)),
                    Flatten(),
                    Dense(512, activation='relu'),
                    Dense(1, activation='sigmoid')
])
```

- **Resumo do Modelo:**
    - 3 blocos `Conv2D + MaxPooling`.
    - Camada densa de 512 neurônios com ReLU.
    - Camada de saída com sigmoid para classificação binária.

⚙️ Compilação do Modelo

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d_6 (MaxPooling2D)	(None, 63, 63, 16)	0
conv2d_7 (Conv2D)	(None, 61, 61, 32)	4,640
max_pooling2d_7 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_8 (Conv2D)	(None, 28, 28, 64)	18,496
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_2 (Flatten)	(None, 12544)	0
dense_4 (Dense)	(None, 512)	6,423,040
dense_5 (Dense)	(None, 1)	513

```
Total params: 6,447,137 (24.59 MB)
Trainable params: 6,447,137 (24.59 MB)
Non-trainable params: 0 (0.00 B)
```

🚀 Treinamento

- Rodado por **10 épocas**:

```
history = model.fit(train_iterator, epochs=10, validation_data=val_iterator)
```

```
Epoch 1/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 117s 3s/step - accuracy: 0.5195 - loss: 0.8399 - val_accuracy: 0.6528 - val_loss: 0.6235
Epoch 2/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.6703 - loss: 0.6057 - val_accuracy: 0.6018 - val_loss: 0.6523
Epoch 3/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 113s 3s/step - accuracy: 0.6072 - loss: 0.6475 - val_accuracy: 0.7214 - val_loss: 0.5492
Epoch 4/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 113s 3s/step - accuracy: 0.7178 - loss: 0.5509 - val_accuracy: 0.7468 - val_loss: 0.5149
Epoch 5/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 111s 3s/step - accuracy: 0.7364 - loss: 0.5285 - val_accuracy: 0.7148 - val_loss: 0.5469
Epoch 6/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 114s 3s/step - accuracy: 0.7352 - loss: 0.5248 - val_accuracy: 0.7686 - val_loss: 0.4839
Epoch 7/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.7584 - loss: 0.4993 - val_accuracy: 0.7880 - val_loss: 0.4549
Epoch 8/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.7717 - loss: 0.4711 - val_accuracy: 0.7886 - val_loss: 0.4448
Epoch 9/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 110s 3s/step - accuracy: 0.7695 - loss: 0.4811 - val_accuracy: 0.7974 - val_loss: 0.4356
Epoch 10/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 113s 3s/step - accuracy: 0.7805 - loss: 0.4556 - val_accuracy: 0.7992 - val_loss: 0.4405
```

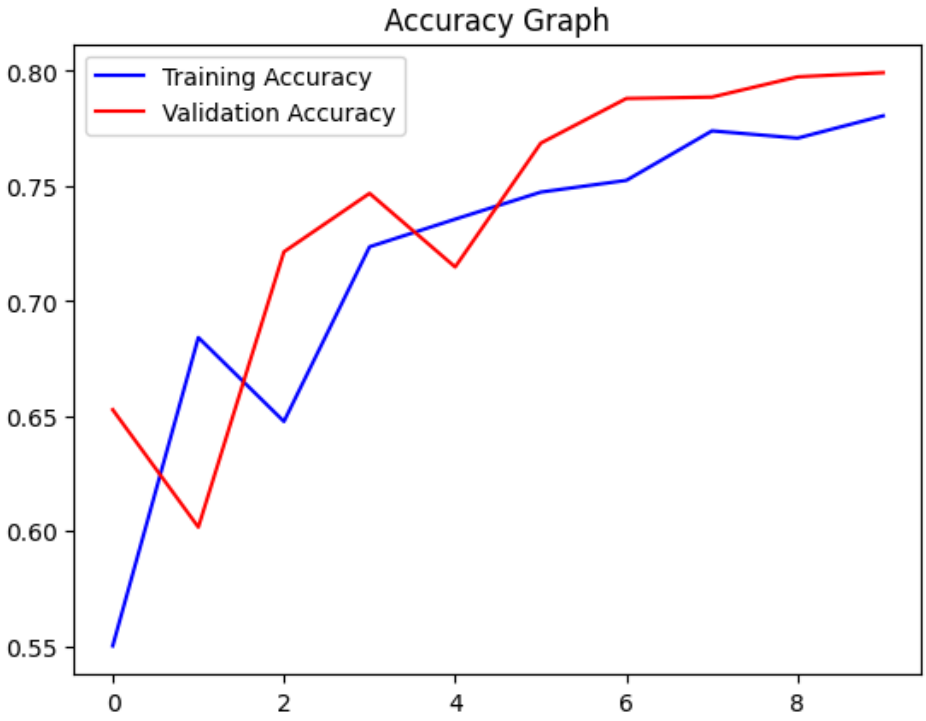
5️⃣ Análise dos Resultados

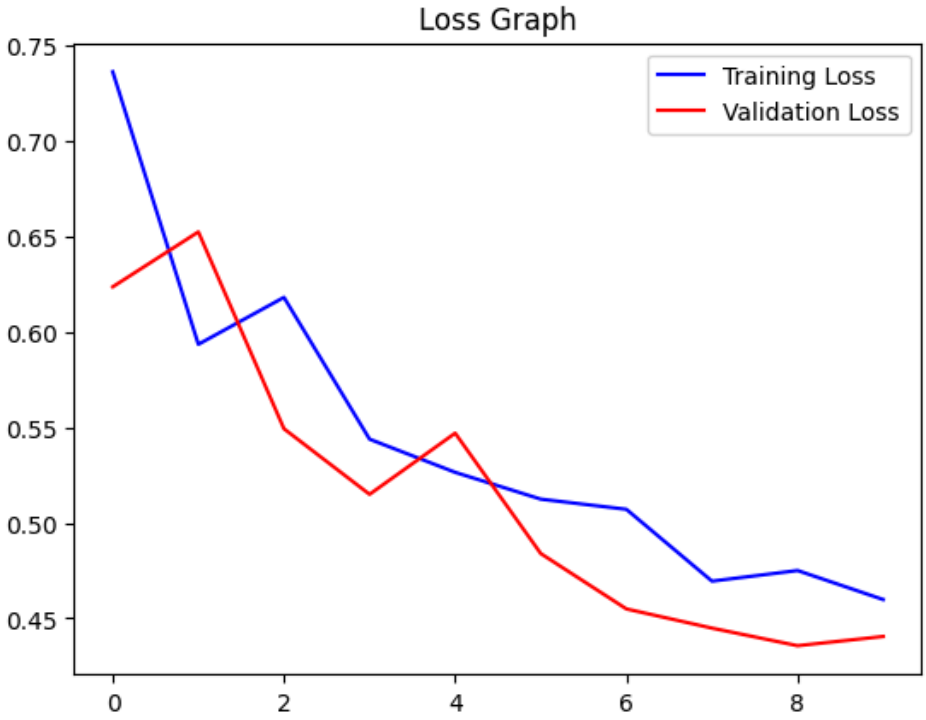
📊 Gráficos de Desempenho

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Accuracy Graph')
plt.legend()
plt.figure()

loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()
```


Accuracy Graph
Training Accuracy
Validation Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0
2
4
6
8


Loss Graph
Training Loss
Validation Loss
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
2
4
6
8

- Os gráficos ajudam a avaliar se há:
    - **Overfitting** (validação piora enquanto treino melhora).
    - **Underfitting** (ambos ruins).
    - **Bom ajuste** (curvas próximas e estáveis).

6️⃣ Teste com Imagem Real

🔍 Código de Inferência

```
image_path = "dog.jpg" # path of the image
img = load_img(image_path, target_size=(128, 128))
img = np.array(img)
img = img / 255.0 # normalize the image
img = img.reshape(1, 128, 128, 3) # reshape for prediction
pred = model.predict(img)
if pred[0] > 0.5:
    label = 'Dog'
else:
    label = 'Cat'
print(label)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Dog
```

- O modelo recebe uma imagem externa e retorna `'Dog'` ou `'Cat'` conforme a previsão.

🧠 Funcionamento Interno da CNN

- **Conv2D:** Extrai características locais (bordas, texturas).
- **MaxPooling:** Reduz dimensionalidade, mantendo características principais.
- **Flatten:** Converte mapas de ativação em um vetor.
- **Dense:** Aprende a combinação dessas características para tomada de decisão.
- **Sigmoid:** Saída probabilística (0 a 1) → se > 0.5 classifica como “Dog”.

🔧 Possíveis Melhorias

- Uso de arquiteturas pré-treinadas (Transfer Learning) como **VGG16, ResNet50**.
- Aumento do tamanho da imagem para capturar mais detalhes.
- Regularização (Dropout, BatchNormalization).
- Mais épocas, ajuste de batch size e learning rate.

✅ Conclusão

Este modelo CNN foi capaz de classificar imagens de cães e gatos com bom desempenho, utilizando uma arquitetura relativamente simples. A utilização de **Data Augmentation** foi fundamental para mitigar overfitting e melhorar a generalização do modelo.