Inteligência Artificial

812839 - Vinícius Miranda de Araújo

Lista de Exercícios 12

```
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras import layers, models
from keras.utils import load_img, img_to_array
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

Base de Dados

- A base de dados "Cats vs Dogs" foi retirada do Kaggle.
- Essa base continha aproximadamente 14.000 imagens de cães e gatos, cada.

Preparação dos Dados

Dowload da Base de Dados

```
import kagglehub

path = kagglehub.dataset_download("shaunthesheep/microsoft-catsvsdogs-dataset")

print("Path to dataset files:", path)
```

Carregando e Pré-Processando as Imagens

```
# Caminhos das imagens
dir_cat = "/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Cat"
dir_dog = "/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Dog"
```

```
# Parâmetros gerais
img_size = (150, 150)  # Tamanho padrão das imagens
batch_size = 32
```

```
# Pré-processamento de imagem: redimensionamento e normalização
def load_images_from_folder(folder, label, max_images=1000):
    images = []
    labels = []
    count = 0
    for filename in os.listdir(folder):
        path = os.path.join(folder, filename)
        try:
            img = load_img(path, target_size=img_size)  # Redimensiona a imagem
            img = img_to_array(img) / 255.0  # Converte para array e normaliza
            images.append(img)
            labels.append(label)
            count += 1
            if count >= max_images:
                break
        except:
            continue  # Ignora imagens corrompidas
    return images, labels
```

```
# Carregando imagens de gatos (0) e cachorros (1)
cat_images, cat_labels = load_images_from_folder(dir_cat, 0)
dog_images, dog_labels = load_images_from_folder(dir_dog, 1)

print(f'Gatos: {len(cat_images)} imagens carregadas.')
print(f'Cachorros: {len(dog_images)} imagens carregadas.')
```

Separando os Dados de Treino, Validação e Teste

```
# Unindo e embaralhando
X = np.array(cat_images + dog_images)
y = np.array(cat_labels + dog_labels)

print(f'Total de imagens: {len(X)}')
print(f'Total de rótulos: {len(y)}')
```

```
# Divisão: treino (70%), validação (15%), teste (15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

print(f'Total de imagens: {len(X)}')
print(f'Total de rótulos: {len(y)}')
print(f'Treino: {len(X_train)} | Validação: {len(X_val)} | Teste: {len(X_test)}')
```

Aumento de dados (data augmentation)

```
# Aumento de dados (data augmentation) para o treino
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

print("Data augmentation configurado para o conjunto de treino.")
```

Construção e Treinamento do Modelo CNN

```
# Arquitetura da CNN
model = models.Sequential([
    layers.Input(shape=(150, 150, 3)),  # Adicione esta linha
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

print("Modelo CNN construído com sucesso.")
model.summary()
```

```
# Compilação do modelo
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

print("Modelo compilado com sucesso.")
```

```
history = model.fit(
    train_generator,
    batch_size=batch_size,
    epochs=10,
    validation_data=val_generator
)

print("Modelo treinado com sucesso.")
```

Avaliação e Testes

Avaliando os Resultados

```
# Gráficos de acurácia e perda
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Acurácia Treino')
plt.plot(history.history['val_accuracy'], label='Acurácia Validação')
plt.title('Acurácia por Época')
plt.xlabel('Época')
plt.ylabel('Acurácia')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Perda Treino')
plt.plot(history.history['val_loss'], label='Perda Validação')
plt.title('Perda por Época')
plt.xlabel('Época')
plt.ylabel('Perda')
plt.legend()

plt.tight_layout()
plt.show()
```

```
# Avaliação no Conjunto de Teste
test_generator.reset()
predictions = model.predict(test_generator)
y_pred = (predictions > 0.5).astype(int).flatten()
y_true = y_test


print("\nRelatório de Classificação:")
print(classification_report(y_true, y_pred, target_names=["Cat", "Dog"]))

print("\nMatriz de Confusão:")
print(confusion_matrix(y_true, y_pred))

print("\nAcurácia no conjunto de teste:", np.mean(y_pred == y_true))
print("Acurácia no conjunto de teste:", model.evaluate(test_generator)[1])
```

Testando com Imagens Diferentes

```
# Função para testar imagem externa
def testar_imagem(path):
    imagem = load_img(path, target_size=img_size)
    imagem_array = img_to_array(imagem) / 255.0
    imagem_array = np.expand_dims(imagem_array, axis=0)
    pred = model.predict(imagem_array)[0][0]
    classe = "Dog" if pred > 0.5 else "Cat"
    print(f"Imagem: {os.path.basename(path)} | Classificação: {classe} ({pred:.2f})")
```

```
print("\nTestando imagens externas:")
testar_imagem("/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Cat/2001.jpg")
testar_imagem("/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Dog/2001.jpg")
testar_imagem("/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Cat/2002.jpg")
testar_imagem("/kaggle/input/microsoft-catsvsdogs-dataset/versions/1/PetImages/Dog/2002.jpg")
```

Conclusão

A rede neural convolucional implementada foi capaz de aprender padrões visuais relevantes e realizar a classificação de forma eficaz. A combinação de pré-processamento, aumento de dados e arquitetura convolucional permitiu um desempenho robusto, mesmo com uma base de dados relativamente simples. Além disso, como a CNN foi treinada com uma quantidade relativamente baixa de instâncias, os resultados obtidos foram abaixo do esperado. Entretando, ao aumentar a quantidade de imagens para próximo de 5.000 imagens de cada classe, a acurácia ficou próxima dos 78%, indicando que a quantidade de imagens é extremamente relevante para a melhora ou piora do modelo.