FIAP

CURSO TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

NOMES: Murillo Ferreira Ramos
Pedro Luiz Prado
William Kenzo Hayashi

Challenge – OdontoPrev

OdontoPrev Resolução de problemas de Redução de Sinistros

São Paulo
-2024-

Murillo Ferreira Ramos
Pedro Luiz Prado
William Kenzo Hayashi

Challenge – OdontoPrev

OdontoPrev Resolução de problemas de Redução de Sinistros

Challenge apresentado pela empresa OdontoPrev, como parte do desenvolvimento das mais diversas áreas de conhecimento do curso.

São Paulo
-2024-

Atualizações do Projeto

Este estágio do projeto configurou o acesso ao Kaggle e utilizou o `Panoramic Dental Dataset` para testes iniciais de classificação de imagens dentárias com redes neurais convolucionais (CNNs). Embora o pipeline de carregamento, pré-processamento, treinamento e avaliação esteja em funcionamento, os resultados ainda não são satisfatórios.

Tanto as imagens quanto o treinamento do modelo necessitam de ajustes para melhorar a precisão. Com isso, estamos revendo o pré-processamento e a arquitetura da rede neural para otimizar o desempenho do modelo.

Ferramentas e Bibliotecas Usadas

- **`os`**: Gerenciamento de arquivos e diretórios, usado para manipular e listar arquivos no sistema.
- **`numpy`**: Operações matemáticas e manipulação de arrays, especialmente útil para processar dados de imagem como arrays.
- **`pandas`**: Leitura e manipulação de dados estruturados, aqui usado para carregar rótulos (labels) em formato CSV.
- **`tensorflow.keras`**: Framework para construir e treinar redes neurais, fornecendo funções para carregar imagens e criar um modelo CNN.
- **`sklearn.model_selection`**: Ferramenta para dividir o dataset em conjuntos de treino e validação.
- **`matplotlib.pyplot`**: Biblioteca de visualização, usada para plotar métricas de desempenho (acurácia) durante o treinamento.

Etapas do Projeto

1. Configuração de Acesso ao Kaggle

Primeiro, configuramos o ambiente para autenticação com a API do Kaggle. Isso requer o arquivo `kaggle.json`, que contém as credenciais de API, para permitir o download direto do dataset.

```
#configurando a autenticação para conseguir acessar o kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

2. Configuração de Acesso ao Kaggle

```
# confirmar se está funcionando o kaggle
!kaggle datasets list
```

Aqui, executamos um teste básico para listar datasets e confirmar que a autenticação e a conexão com o Kaggle estão funcionando corretamente.

3. Download do Dataset

Usamos o comando abaixo para baixar o `Panoramic Dental Dataset`, que contém imagens dentárias panorâmicas para análise.

```
# confirmar se está funcionando o kaggle
!kaggle datasets list
```

4. Descompactação do Dataset

Descompactamos o arquivo `.zip` do dataset para acessar os arquivos de imagem e rótulos.

```
#descompactando dataset
!unzip panoramic-dental-dataset.zip -d ./datasets
```

5. Listagem de Imagens

Com a biblioteca `os`, listamos os primeiros arquivos do diretório de imagens para verificar se foram carregados corretamente.

```
import os

# Definindo o caminho correto para as imagens
image_dir = '/content/datasets/images'
image_files = os.listdir(image_dir)

# Exibindo os primeiros 10 arquivos de imagem
print(image_files[:10])
```

6. Verificação dos Arquivos de Rótulos

Caso o dataset contenha rótulos em um diretório específico, verificamos a existência desse diretório e listamos os arquivos para garantir que todos os dados estejam acessíveis.

```
[ ]  # Definindo o caminho para os rótulos
     label_dir = '/content/datasets/labels'  # ou '/content/datasets/labels_cut'
     if os.path.exists(label_dir):
         label_files = os.listdir(label_dir)
         print("Arquivos de rótulos:", label_files)
     else:
         print("Diretório de rótulos não encontrado.")
```

7. Pré-processamento das Imagens

Utilizamos `tensorflow.keras.preprocessing.image` para carregar e redimensionar as imagens para 224x224 pixels. Em seguida, normalizamos os valores dos pixels para um intervalo de 0 a 1, o que ajuda a estabilizar o treinamento do modelo.

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

def preprocess_image(image_path, target_size=(224, 224)):
    img = load_img(image_path, target_size=target_size)
    img_array = img_to_array(img)
    img_array /= 255.0  # Normalizando os valores dos pixels
    return img_array

# Carregar todas as imagens
image_data = []
for img_file in image_files:
    img_path = os.path.join(image_dir, img_file)
    image_data.append(preprocess_image(img_path))

# Converter para um array numpy
image_data = np.array(image_data)
print(f"Imagens carregadas: {image_data.shape}")
```

8. Carregamento dos Rótulos

Utilizamos `pandas` para carregar o arquivo CSV de rótulos, caso esteja disponível, e exibimos as primeiras linhas para verificar o conteúdo.

```python
import pandas as pd

# Se houver um arquivo CSV com rótulos
label_file = '/content/datasets/labels/labels.csv'
if os.path.exists(label_file):
    labels_df = pd.read_csv(label_file)
    print(labels_df.head())
else:
    print("Arquivo de rótulos CSV não encontrado.")
```

9. Divisão do Dataset

Usamos a função `train_test_split` da biblioteca `sklearn.model_selection` para dividir o dataset em conjunto de treino (80%) e validação (20%), o que ajuda a evitar overfitting.

```python
from sklearn.model_selection import train_test_split


labels = np.array([0 if 'normal' in img else 1 for img in image_files])

X_train, X_val, y_train, y_val = train_test_split(image_data, labels, test_size=0.2, random_state=42)

print(f"Imagens de treino: {X_train.shape}, Rótulos de treino: {y_train.shape}")
print(f"Imagens de validação: {X_val.shape}, Rótulos de validação: {y_val.shape}")
```

10. Construção do Modelo CNN

Criamos uma rede neural convolucional (CNN) com `tensorflow.keras`, composta por camadas de convolução, pooling e camadas densas. O modelo é configurado para classificação binária (0 ou 1).

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Criando o modelo de classificação
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D((2, 2)))

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

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))  # Para classificação binária

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

# Resumo do modelo
model.summary()
```

11. Treinamento do Modelo

Executamos o treinamento do modelo por 10 épocas, monitorando o desempenho com o conjunto de validação.

```
[ ]  # Treinando o modelo
     treinamento = model.fit(X_train, y_train, epochs=10, validation_d
```

12. Avaliação do Modelo

Avaliamos o modelo com o conjunto de validação para verificar a acurácia final.

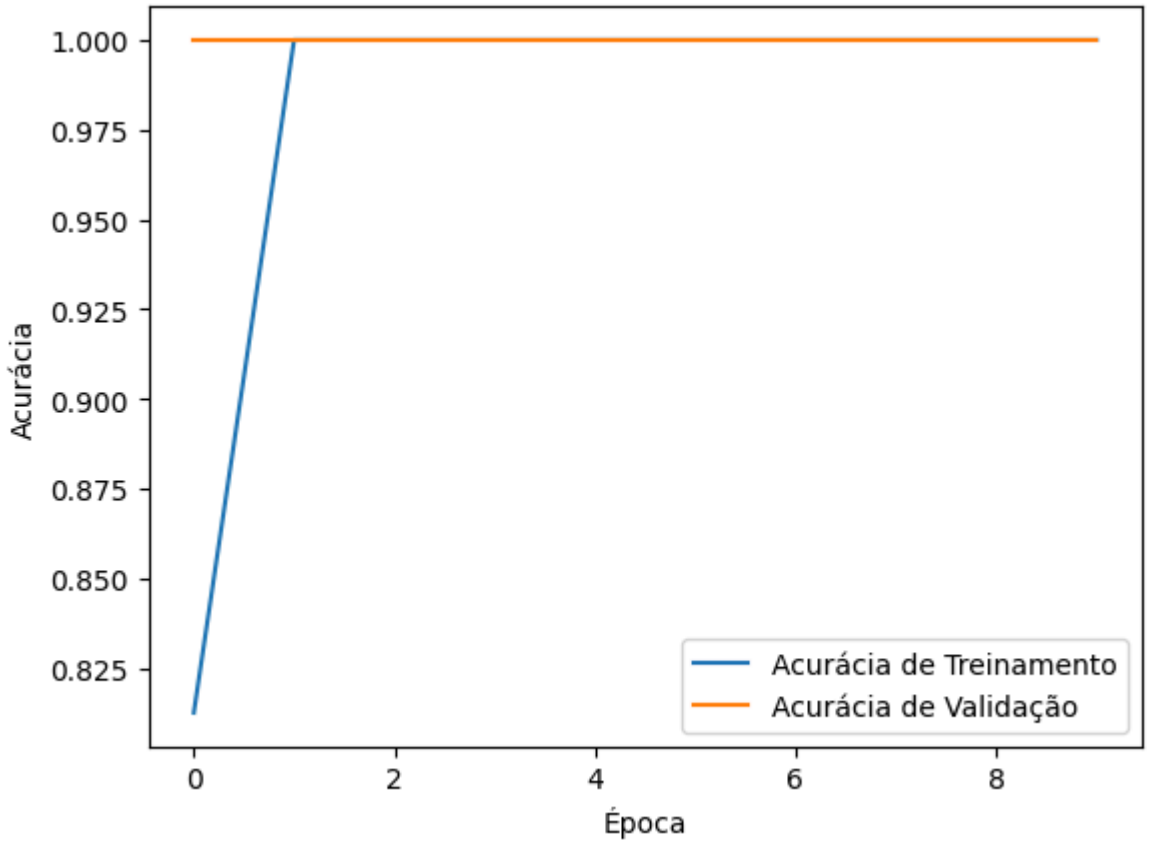
```python
# Avaliar o modelo no conjunto de validação
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Acurácia no conjunto de validação: {accuracy:.2f}")
```

12. Visualização do Desempenho

Usamos `matplotlib.pyplot` para plotar a acurácia durante o treinamento e verificar a evolução da performance.

```python
import matplotlib.pyplot as plt

# Plotar a acurácia e perda ao longo do treinamento
plt.plot(treinamento.history['accuracy'], label='Acurácia de Treinamento')
plt.plot(treinamento.history['val_accuracy'], label='Acurácia de Validação')
plt.xlabel('Época')
plt.ylabel('Acurácia')
plt.legend()
plt.show()
```



Resumo do Desenvolvimento

O projeto utiliza o `Panoramic Dental Dataset` do Kaggle para treinar uma rede neural convolucional (CNN) com o objetivo de classificar imagens dentárias. As etapas realizadas incluem configuração do ambiente, download e pré-processamento do dataset, construção e treinamento do modelo CNN, e análise dos resultados.

Apesar de o pipeline de processamento estar operacional, os resultados ainda não atingiram o desempenho desejado. A acurácia do modelo e a qualidade das previsões ainda precisam ser aprimoradas, possivelmente por meio de ajustes no pré-processamento das imagens e na arquitetura da rede neural. Essas melhorias visam aumentar a precisão e a capacidade de generalização do modelo para que ele possa fornecer classificações mais confiáveis no futuro.