UNIVERSIDADE DO MINDELO DEPARTAMENTO

DE ENGENHARIA E RECURSOS DO MAR

CURSO DE LICENCIATURA EM

ENGENHARIA INFORMÁTICA E SISTEMAS COMPUTACIONAIS

ATIVIDADE PRÁTICA

ANO LETIVO 2023/2024 – 4ºANO

TEMA: Sistema de Classificação de Alunos em Tempo Real com TransferLearning.

Discente: Ruben Ferreiro Nº:5383

Docente: Steven Fortes

Mindelo, 2024

Índice

Conteúdo

Objetivo

O objetivo deste projeto é desenvolver um sistema para detenção e classificação de rostos em tempo real usando técnicas de deep learning. O projeto envolve o treinamento de um modelo de rede neural convolucional para distinguir entre diferentes classes de rostos, como estudantes e não estudantes, utilizando transferência de aprendizado do modelo VGG16. Este modelo treinado é então integrado em uma aplicação web que captura vídeo de uma webcam, processa os quadros de vídeo para detetar rostos e exibe os resultados da classificação em tempo real.

Tecnologias Utilizadas:

- **Flask**: Framework web em Python usado para criar a aplicação web.
- **OpenCV**: Biblioteca de visão computacional usada para capturar e processar os quadros de vídeo da webcam.
- **TensorFlow/Keras**: Frameworks de deep learning usados para treinar e carregar o modelo de rede neural convolucional.
- **VGG16**: Modelo pré-treinado de rede neural convolucional utilizado como base para transferência de aprendizado.
- **Numpy**: Biblioteca para operações numéricas, usada principalmente para manipulação de arrays.
- **Matplotlib**: Biblioteca de visualização de dados usada para plotar gráficos de acurácia e perda durante o treinamento do modelo.

Estrutura do Projeto:

Train_Model.py

Relacionados ao modelo utilizado

Esta seção do código está relacionada ao carregamento do modelo base e então adiciona algumas camadas a ele para torná-lo mais preciso e, após isso, o compila novamente.

```
# Load base model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Add custom layers
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Freeze base model layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

Relacionado aos dados

Esta parte é responsável pela geração de dados e pelo treinamento do modelo. Após o treinamento, ele irá salvar o modelo no arquivo.

```
# Prepare data generator
train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, zoom_range=0.2, shear_range=0.2, rotation_range=40)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=10
)

# Save the trained model
model.save('your_model.h5')
```

Plot the data

Esta parte é para plotar os gráficos que mostram a precisão do modelo utilizado.

```
# Plots to explain
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()
```

App.py

Função predict_face

Esta função recebe um frame de vídeo, processa a imagem e utiliza o modelo carregado para fazer a predição.

```python
# Function to predict face using the loaded model
def predict_face(frame):
    # Preprocess the frame for prediction
    face = cv2.resize(frame, (224, 224))  # Resize to match the input shape of the model
    face = img_to_array(face)
    face = np.expand_dims(face, axis=0)
    face = face / 255.0  # Normalize

    prediction = model.predict(face)[0][0]

    return prediction

```

Função real_time_detection

Esta função inicia a deteção de rostos em tempo real.

```python
# Route for real-time face detection
@app.route('/real_time_detection')
def real_time_detection():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

```

Função generate_frames

Esta função captura frames da webcam, processa cada frame para detecção de rostos e transmite os frames processados.

```python
# Function to generate frames for real-time detection
def generate_frames():
    camera = cv2.VideoCapture(0)
    while True:
        success, frame = camera.read()
        if not success:
            break
        else:
            # Use the prediction function
            prediction = predict_face(frame)
            label = 'Student' if prediction > 0.5 else 'Others'

            # Display the label on the frame
            cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
    camera.release()
```

Desafios Enfrentados:

1. Preparação e Qualidade dos Dados

A eficácia do modelo depende da qualidade e diversidade dos dados de treinamento. Dados mal rotulados ou não representativos podem levar a um modelo com baixo desempenho. O aumento de dados usando técnicas como o ImageDataGenerator pode introduzir variabilidade, mas parâmetros inadequados podem adicionar ruído em vez de valor.

2. Desempenho em Tempo Real

A aplicação envolve processamento de vídeo em tempo real, o que pode introduzir latência significativa. A leitura contínua da câmera e o processamento de cada quadro podem afetar a fluidez da deteção. Além disso, o uso intensivo de recursos de CPU e memória pode ser um desafio, especialmente em ambientes com recursos limitados.

3. Integração e Compatibilidade

Integrar o modelo treinado com a aplicação web pode ser um desafio, especialmente em termos de compatibilidade de versões das bibliotecas usadas para treinamento e aquelas usadas na aplicação. Por exemplo, diferenças nas versões do TensorFlow ou Keras podem causar problemas ao carregar o modelo. Garantir que o ambiente de desenvolvimento e o ambiente de produção estejam alinhados é essencial para uma integração suave.

Passos Futuros:

1. Adicionar Botão Invisível para Acesso ao Perfil do Aluno

Adicionar um botão na interface que permanece invisível até que um aluno seja reconhecido. Após o reconhecimento, o botão se torna visível, permitindo que o usuário seja redirecionado ao perfil do aluno reconhecido.

2. Escalar para Reconhecer Mais Alunos e Gerenciar Perfis

Coletar e rotular mais dados de diferentes alunos para treinar o modelo com maior diversidade. Implementar uma base de dados para armazenar e gerenciar os perfis dos alunos reconhecidos, incluindo informações detalhadas e personalizadas.

3. Melhoria da Qualidade dos Dados

Garantir a precisão e a consistência das etiquetas dos dados. Utilizar técnicas de aumento de dados para introduzir variabilidade e melhorar a robustez do modelo.

4. Otimização do Modelo

Explorar outras arquiteturas de modelos pré-treinados e ajustar finamente as camadas do modelo. Implementar validação cruzada para avaliar melhor o desempenho do modelo e evitar overfitting.

Referências:

- https://www.youtube.com/watch?v=mjk4vDYOwq0&ab_channel=CodeWithAarohi
- https://www.youtube.com/watch?v=o3bCVqF9gI0&list=PL7yh-TELLS1EyAye_UMnlsTGKxg8uatkM&ab_channel=NeuralNine
- https://www.youtube.com/watch?v=ZkZuVQ-vyv0&ab_channel=KrishNaik
- https://www.youtube.com/watch?v=i_-m1kBTdBI&ab_channel=KNOWLEDGEDOCTOR