UNIVERSIDADE DO MINDELO
DEPARTAMENTO DE ENGENHARIA E CIÊNCIAS DO MAR

CURSO DE LICENCIATURA EM ENGENHARIA INFORMÁTICA E SISTEMA COMPUTACIONAIS

Relatório

Tema: Relatório de Projeto Final

Autor: Natanael Duarte, Nº5073

Professor: Steven Fortes

Mindelo, 2024

UNIVERSIDADE DO MINDELO
DEPARTAMENTO DE ENGENHARIA E CIÊNCIAS DO MAR

CURSO DE LICENCIATURA EM
ENGENHARIA INFORMÁTICA E SISTEMAS COMPUTACIONAIS

RELATÓRIO

Tema: Relatório de Projeto Final

Autor: Natanael Duarte, Nº 5073

Professor: Steven Fortes

Mindelo, 2024

Índice

INTRODUÇÃO

Este relatório detalha o desenvolvimento de um projeto utilizando Flask para realizar reconhecimento facial em tempo real, integrando um modelo de rede neural convolucional (CNN) treinado previamente. O projeto envolveu várias etapas, incluindo o treinamento do modelo no Google Colab, a integração do modelo treinado no Flask e a exibição do vídeo da webcam em uma página web.

1. TREINAMENTO DO MODELO NO GOOGLE COLAB

O treinamento do modelo foi realizado no Google Colab devido à capacidade computacional oferecida e à facilidade de integração com o TensorFlow. Os passos principais realizados durante esta fase foram os seguintes:

- Preparação dos Dados: Utilização de um conjunto de dados contendo imagens de faces para treino e validação. As imagens foram preparadas e organizadas em subconjuntos de treino e teste.

- Desenvolvimento da Arquitetura do Modelo: Implementação de uma CNN (Convolutional Neural Network) utilizando TensorFlow e Keras. A arquitetura do modelo foi escolhida com base na tarefa de reconhecimento facial em tempo real.

- Treinamento e Ajuste de Hiperparâmetros: Treinamento do modelo utilizando técnicas de aumento de dados para aumentar a diversidade do conjunto de treinamento. Foram ajustados hiperparâmetros como taxa de aprendizado, tamanho do lote (batch size) e número de épocas de treinamento para otimizar o desempenho do modelo.

- Avaliação do Modelo: Avaliação da precisão e da perda do modelo utilizando o conjunto de validação. Métricas como acurácia e matriz de confusão foram utilizadas

para avaliar a capacidade do modelo de reconhecer corretamente faces em imagens não vistas durante o treinamento.

2. INTEGRAÇÃO DO MODELO NO PROJETO FLASK

Após o treinamento e a validação do modelo no Google Colab, o próximo passo foi integrar o modelo no projeto Flask. Isso envolveu:

- Exportação e Salvamento do Modelo: Salvamento do modelo treinado em um formato compatível com o TensorFlow/Keras, como um arquivo HDF5 (`model_natanael_recognition_TL.h5`). Este arquivo contém a arquitetura da rede neural, os pesos treinados e outras configurações necessárias para realizar previsões.

- Carregamento do Modelo no Flask: Utilização da biblioteca TensorFlow/Keras para carregar o modelo treinado dentro do código Flask. Isso foi realizado na rota específica do Flask que realiza a classificação em tempo real das imagens capturadas pela webcam.

- Processamento de Imagens em Tempo Real: Implementação de uma função para processar cada quadro capturado pela webcam em tempo real. Esta função aplica o modelo carregado para realizar a classificação das faces presentes na imagem, gerando previsões que são então exibidas na interface web.

3. EXPLICAÇÃO DETALHADA DO CODIGO

1. Importação das bibliotecas necessárias:
   - `Flask`: para a criação do aplicativo web.
   - `request`, `render_template`, `Response`: para lidar com solicitações HTTP, renderizar templates HTML e fornecer respostas.
   - `cv2`: para processamento de imagens e captura de vídeo.
   - `numpy`: para operações matemáticas e manipulação de matrizes.
   - `tensorflow`: para a criação e treinamento de modelos de aprendizado de máquina.

2. Criação de uma instância do Flask:

- `app = Flask(__name__)`

3. Definição do caminho para o modelo pré-treinado:

- `model_path = 'model_natanael_recognition_TL.h5'`

4. Tentativa de carregar o modelo pré-treinado:

- `try:`: bloco de código que tenta executar a ação.

- `model = tf.keras.models.load_model(model_path)`: carrega o modelo pré-treinado.

- `model.summary()`: exibe uma descrição do modelo.

- `except Exception as e:`: bloco de código que é executado se ocorrer uma exceção durante a tentativa de carregamento do modelo.

- `print(f"Error loading model: {e}")`: imprime a mensagem de erro com a exceção.

5. Função para processar um quadro e classificar a face:

- `def process_frame(frame):`

- `img = cv2.resize(frame, (220, 220))`: redimensiona a imagem para o tamanho esperado pelo modelo.

- `img = img / 255.0`: normaliza os valores dos pixels para o intervalo [0, 1].

- `img = np.expand_dims(img, axis=3)`: adiciona uma dimensão extra ao array (necessário para o modelo).

- `predictions = model.predict(img)`: faz a predição usando o modelo.

- `return predictions`: retorna as predições.

6. Função para capturar quadros da câmera e classificar em tempo real:

- `def gen_frames():`

- `print("Starting video capture")`: imprime uma mensagem de início.

- `camera = cv2.VideoCapture(0)`: abre a câmera.

- `if not camera.isOpened():`: verifica se a câmera foi aberta com sucesso.

- `print("Error: Could not access the webcam.")`: imprime uma mensagem de erro se a câmera não puder ser acessada.

- `while True:`: loop infinito para capturar e processar quadros.

- `success, frame = camera.read()`: lê um quadro da câmera.

- `if not success:`: verifica se um quadro foi lido com sucesso.

- `print("Failed to read frame")`: imprime uma mensagem de erro se um quadro não puder ser lido.

- `break`: sai do loop infinito.

- `else:`: bloco de código que é executado se um quadro for lido com sucesso.

- `predictions = process_frame(frame)`: processa o quadro e obtém as predições.

- `text = f'Prediction: {"Natanael" if np.argmax(predictions) >= 0.5 else "outros"}'`: cria uma string com a predição formatada.

- `cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)`: escreve a predição no quadro.

- `ret, buffer = cv2.imencode('.jpg', frame)`: codifica o quadro em JPEG.

- `frame = buffer.tobytes()`: converte o buffer para bytes.

- `yield (b'--frame\r\n' + b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')`: produz um quadro para ser enviado ao cliente.

- `camera.release()`: libera a câmera.

- `print("Released video capture")`: imprime uma mensagem de término.

7. Definição das rotas do aplicativo:

- `@app.route('/')`: define a rota para a página inicial.

- `@app.route('/about')`: define a rota para a página sobre.

- `@app.route('/model')`: define a rota para a página do modelo.

- `@app.route('/webcam')`: define a rota para a página da câmera.

- `@app.route('/video_feed')`: define a rota para o feed de vídeo.

8. Execução do aplicativo Flask:

- `if __name__ == '__main__':`

- `app.run(debug=True)`: inicia o aplicativo Flask com o depurador ativado.

4. EXIBIÇÃO DO VÍDEO DA WEBCAM NA PÁGINA WEB

A exibição do vídeo da webcam na página web foi uma parte crucial do projeto, permitindo a interação do usuário com o sistema de reconhecimento facial. Para isso, foram realizados os seguintes passos:

- Configuração da Rota de Vídeo: Implementação de uma rota no Flask (`/video_feed`) que gera um stream de vídeo utilizando a biblioteca OpenCV. Esta rota é responsável por capturar os quadros da webcam e transmiti-los como uma resposta HTTP multipart para a página web.

- Interface HTML e CSS: Desenvolvimento de uma página HTML (`webcam.html`) que utiliza HTML5 e JavaScript para exibir dinamicamente o vídeo da webcam. Um arquivo CSS (`styles.css`) foi utilizado para estilizar a página e garantir uma apresentação profissional e responsiva.

DESAFIOS ENFRENTADOS

Durante o desenvolvimento do projeto, diversos desafios foram identificados e superados:

-Compatibilidade e Versões: Garantir que o modelo treinado no Google Colab fosse compatível com as versões de bibliotecas utilizadas no ambiente Flask, como TensorFlow e OpenCV. Isso envolveu atualizações e ajustes nas dependências do projeto.

-Otimização de Desempenho: Lidar com o desempenho do modelo em tempo real, garantindo que a classificação das faces fosse realizada de forma eficiente e sem grandes atrasos perceptíveis pelo usuário.

-Integração de Componentes: Integrar efetivamente o streaming de vídeo da webcam com o processamento de imagem em tempo real e a exibição das previsões na interface web. Isso exigiu um bom entendimento das tecnologias envolvidas e a resolução de problemas técnicos específicos ao longo do desenvolvimento.

CONCLUSÃO

O projeto de reconhecimento facial em tempo real com Flask e TensorFlow foi concluído com sucesso, demonstrando a integração eficaz de um modelo de rede neural treinado com uma aplicação web interativa. A utilização do Google Colab facilitou o treinamento do modelo, enquanto o Flask permitiu a criação de uma interface web responsiva e funcional.

Este relatório cobriu detalhadamente as etapas de desenvolvimento, os desafios enfrentados e as soluções adotadas ao longo do projeto. A experiência adquirida pode ser aplicada em diversas áreas, como segurança, análise de comportamento do usuário e interação humano-computador baseada em reconhecimento facial.