Yoga Pose Detection Web Application - Deployment Guide

This guide will help you deploy your yoga pose detection system as a web application. The implementation uses Flask to create a web server that processes webcam input, applies your pose detection model, and streams the results back to the browser.

Project Structure

```
yoga_pose_detection/
├── app.py                 # Main Flask application
├── templates/
│   └── index.html         # Web interface template
├── model/
│   ├── model.h5           # Your trained yoga pose model
│   └── labels.npy         # Labels for yoga poses
├── requirements.txt       # Python dependencies
└── README.md              # Project documentation
```

Prerequisites

- Python 3.7+
- Webcam access
- Your trained yoga pose detection model (`model.h5`)
- Labels file (`labels.npy`)

Setup Instructions

1. **Create Project Directory**

   bash

   ```bash
   mkdir -p yoga_pose_detection/templates
   mkdir -p yoga_pose_detection/model
   cd yoga_pose_detection
   ```

2. **Copy Files**
   - Copy the `app.py` file to the root directory
   - Copy `index.html` to the `templates` directory
   - Copy your model files:

     bash

     ```bash
     cp /path/to/your/model.h5 model/
     cp /path/to/your/labels.npy model/
     ```

3. **Create Requirements File** Create a `requirements.txt` file with the following content:

```
flask==2.0.1
opencv-python==4.5.3.56
numpy==1.21.2
mediapipe==0.8.7.3
tensorflow==2.6.0
```

4. **Create Virtual Environment and Install Dependencies**

bash

```bash
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

Running the Application

1. **Start the Flask Server**

bash

```bash
python app.py
```

2. **Access the Web Interface** Open your browser and navigate to:

```
http://localhost:5000
```

3. **Use the Application**
   - Click "Start Detection" to begin webcam streaming and pose analysis
   - Position yourself in the camera frame
   - Perform yoga poses
   - Click "Stop Detection" when finished

Deployment Options

Local Development Server

The instructions above will run the application on a local development server, which is suitable for testing but not for production use.

Production Deployment

For a production environment, consider these options:

Option 1: Deploy with Gunicorn (Linux/Mac)

```bash
pip install gunicorn
gunicorn -w 1 -b 0.0.0.0:8000 app:app
```

Option 2: Deploy on a Cloud Platform

Services like Heroku, AWS Elastic Beanstalk, or Google Cloud Run can host your application. Note that you'll need to ensure the platform supports webcam access, which can be challenging for some cloud environments.

Option 3: Deploy as a Docker Container

Create a `Dockerfile`:

```dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

Build and run the Docker container:

```bash
docker build -t yoga-pose-detection .
docker run -p 5000:5000 yoga-pose-detection
```

Troubleshooting

Camera Access Issues

- Make sure your browser allows camera access
- On some systems, you may need to run the application with administrator/sudo privileges
- If deploying to a server, ensure it has access to a camera or modify the code to accept video uploads instead

Model Loading Issues

- Ensure the paths to `model.h5` and `labels.npy` are correct
- If using Docker, verify the files are included in the Docker image
- Check for TensorFlow version compatibility with your model

Performance Considerations

- The application streams video frames which can be bandwidth-intensive
- Consider reducing resolution or frame rate for better performance
- For high-traffic applications, you might need to implement proper scaling

Customization

Changing the Detection Threshold

- In `app.py`, modify the confidence threshold (currently set to 0.7):

  ```python
  if conf > 0.7:  # Change this value to adjust sensitivity
  ```

Using a Different Camera

- If you have multiple cameras, change the camera index:

  ```python
  cap = cv2.VideoCapture(0)  # Change 0 to 1, 2, etc.
  ```

Adjusting Video Quality

- Modify the frame encoding parameters in the `generate_frames` function:

  ```python
  _, buffer = cv2.imencode('.jpg', frm, [cv2.IMWRITE_JPEG_QUALITY, 85])
  ```

Important Notes

1. **Privacy and Security**: This application accesses the user's camera. Ensure you have appropriate privacy policies and security measures in place.
2. **Browser Compatibility**: WebRTC (used for camera access) is supported by most modern browsers, but there may be variations in performance.
3. **Resource Usage**: Real-time pose detection can be CPU-intensive. Monitor your server's resource usage.

4. **Error Handling**: The provided code includes basic error handling. In a production environment, you might want to add more comprehensive error management.