Contribution report for project

SCENE-DETECT

Farrukh Rasool, f.rasool@oth-aw.de[1]

[1] *OTH Amberg-Weiden, Study programme "Industrie 4.0 Informatik"*

Report written on May 18, 2025

Abstract

The project targets to smart home intelligent system. This syetem is designed to automate, monitor, and assist daily activities, often controlled via sensors, or mobile devices.
this report A smart home intelligent system is designed to automate, monitor, and assist daily activities, often controlled via sensors, or mobile devices. However, for blind or visually impaired individuals current smart home solutions lacks the capability to detect critical physical situations, such as a person lying on the floor or being immobile. A scene-detection system contributes a practical and extendable method for enhancing smart home safety systems, particularly for visually impaired individuals.

Keywords: *scene-detect, smart home, automate*

1. Material

In this project we have used the [1]MediaPipe library along with a comprehensive Python libraries to develop a 3-D coordinate system of dection of persons. The core technologies included [4]OpenCV, [2] TensorFlow and [3]Keras for building and training deep learning models, and NumPy for efficient numerical operations. MediaPipe provided pre-built solutions for human pose and face detection, by creating outline the body postion and enabling accurate person identification in various home environments. This project employed models and layers from Keras for data augmentation to layering model generalization, while matplotlib helped visualize the results and performance metrics. The system processed input from images, detected human presence through pose estimation, and classified scenes using a custom CNN architecture. The combination of these technologies created an assistive system that improves surrounding awareness for visually impaired users through real-time person identification and scene understanding in smart home settings.

2. Method

The model training and prediction pipeline consists of several key steps to classify scenes for smart home assistance for blind individuals. First, an input image (e.g., "Chilling.jpeg") is loaded and preprocessed to match the model's expected input dimensions. The trained model then predicts the class probabilities, with the highest confidence score determining the final classification (e.g., "OK" or "NOT OK", "NOT IN RANGE"). The predicted label and confidence percentage are displayed on the original image using OpenCV, with a green or red text overlay indicating safe or unsafe scenarios, respectively. A white background rectangle ensures text visibility, and the result is visualized using Matplotlib. This pipeline integrates computer vision and deep learning to provide real-time, interpretable feedback for assistive applications.

The system utilizes [1]MediaPipe for real-time person detection and posture analysis. First, an input image (e.g., "Chilling.jpeg") is processed using MediaPipe Pose, which detects human figures and draws a green bounding box around each person. Next, the pipeline extracts 3D skeletal coordinates from the detected pose, including key joints such as shoulders, hips, knees, and ankles. By analyzing the spatial relationships between these points—such as vertical alignment and joint angles—the system determines whether the person is standing upright or lying down. For instance, if the vertical distance between the head and feet exceeds a threshold, it classifies the posture as "NOT OK" , whereas a normal standing pose is labeled "OK".

3. Database concept

This project works for the live data processing so no relational and non-relational (NoSQL) is used whereas for training the model on data set, it follows an organized directory structure to manage training and testing data for the scene detection system:

- **SITUATION/**
  Contains two main subdirectories:
  - **TRAIN/** (for training data)
  - **TEST/** (for evaluation data)
- **TRAIN/ and TEST/ Subfolders:**
  - **OK/**: Images where the scene is safe (e.g., a person standing normally)
  - **NOT OK/**: Images where the scene requires attention (e.g., a person lying down)
  - **NOT IN RANGE/**: Images where no person is detected or the scene is out of bounds

This structure ensures systematic data segregation for model training and validation, enabling efficient classification of scenes in smart home environments.

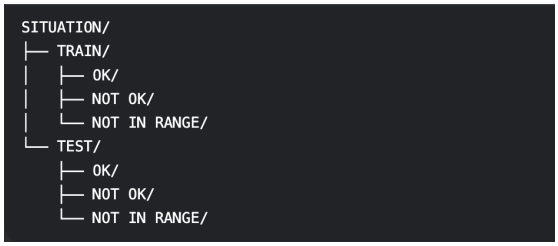


Figure 1. Example figure

Media pipe is a pretrained retrained model and therefore no explicit database is integrated with project. This api works for the live image processing, it follows an organized directory structure to manage training and testing data for the scene detection system: For media pipe, logic used to determine the posture of the person can be shown as follows.

$$\text{Posture} = \begin{cases} \text{Standing (OK)} & \text{if } \text{frame}_{\text{length}} > \text{frame}_{\text{width}} \\ \text{Lying (NOT OK)} & \text{otherwise} \end{cases}$$

4. Implementation aspects

4.1. Convolutional Neural Network (CNN)

The implementation phase focused on constructing a Convolutional Neural Network (CNN) for multi-class image classification using TensorFlow and Keras.The dataset, titled SITUATION.zip, was hosted

on Google Drive and downloaded using the gdown package. The augmentation parameters applied were: Rescaling pixel values to the [0,1] range Random rotations up to 30 degrees Horizontal flipping Zoom and shear transformations These augmentations helped improve the generalization ability of the model by simulating diverse real-world scenarios.

A sequential CNN model was built using the following layers:
Convolutional Layer (32 filters, 3×3)
MaxPooling Layer (2×2) to reduce spatial dimensions
Convolutional Layer (64 filters) + MaxPooling
Convolutional Layer (128 filters) + MaxPooling
Flatten Layer to convert feature maps into a vector
Dense Layer (512 neurons) with ReLU activation
Output Layer (3 neurons) with Softmax activation for 3 predictions
This architecture balances depth and simplicity, allowing effective learning without overfitting, especially suitable for small to medium-sized image datasets.

4.2. MediaPipe

In this implementation, MediaPipe's builtin solution is used to detect the person presence in the picture. The pipeline consists of:
Input Capture using OpenCV for real-time webcam frames
Preprocess handled image internally by MediaPipe, including image normalization and landmark scaling
Draws a green outline around the person body to determine the posture.
Mapping the 3-D points of person from head to foot.
MediaPipe's architecture includes graph-based processing with GPU acceleration, allowing smooth inference even on devices with limited resources.

4.3. Aspect number 2

4.4. Inserting Code

For CNN training we have used Google Colab and used their resources for GPUs , RAM and other resources. For this, we have followed the following steps for our model training:

```
model=models.Sequential()  # Sequential layering.
model.add(layers.Conv2D(32,(3,3),activation='relu')) # First conv layer with 32 filters
model.add(layers.MaxPooling2D((2,2))) # Downsample feature maps
model.add(layers.Conv2D(64,(3,3),activation='relu')) # Second conv layer
model.add(layers.MaxPooling2D((2,2))) # Pooling again
model.add(layers.Conv2D(128,(3,3),activation='relu')) # Third conv layer
model.add(layers.MaxPooling2D((2,2))) # Final pooling
model.add(layers.Flatten()) # Flatten the 3D feature maps to 1D
model.add(layers.Dense(512,activation='relu')) # Fully connected layer with 512 units
model.add(layers.Dense(3,activation='softmax')) # Output layer with 3 neurons (for 3 classes)
```

Code 1. Example of Colab code.

We have used Media pipe library as follows for extracting . For this, we have followed the following steps:

```
# Draw all landmarks and connections
connections = mp_pose. POSE_CONNECTIONS
for connection in connections:
start_idx, end_idx = connection
if landmarks[start_idx].visibility > 0.5 and landmarks[endidx]-visibility > 0.5:
x_start = int(landmarks[start_idxJ-x * scale + offset_x)
y_start = int (landmarks[start_idx].y * scale + offset_y)
x_end = int(landmarks[end_idxJ-x * scale + offset_x)
y_end = int(landmarks[end_idx]-y * scale + offset y)
cv2. line(skeleton_canvas, (x_start, _start), (x_end, y_end), (0, 255, 0), 2)
```

Code 2. Example of Colab code.

Insert code with caution. The code must be self explanatory or it must be explained in the text. Only show important parts of your code.

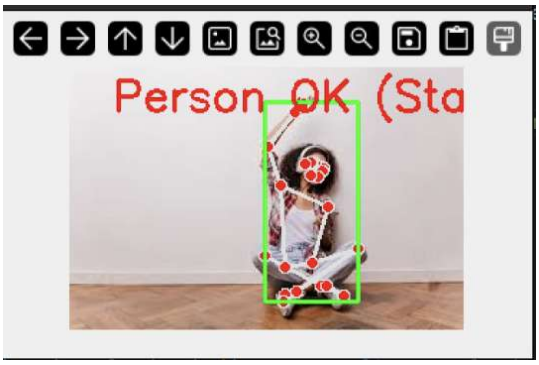
5. Testing

CNN: After training the model on the labeled images of three classes — OK, NOT OK, and NOT IN RANGE.The test dataset is organized in the same directory structure as the training data but contains distinct images that were not used during training. This separation ensures that the model's performance is evaluated on truly new data, offering a realistic measure of its predictive strength. Before testing, all test images undergo pre-processing. This ensures that the input format during testing is consistent with the training phase. Accuracy: The percentage of test images the model correctly classified. Loss: A measure of how confident or uncertain the model was during its predictions. Meida Pipe: For media pipe, the images are tested from instant data source and detect the persons in postions Standing, Sitting and Lying down correctly but in this there is edge case countered of images having persons in upside down postion. In this scenario, it show opposite response as expected.

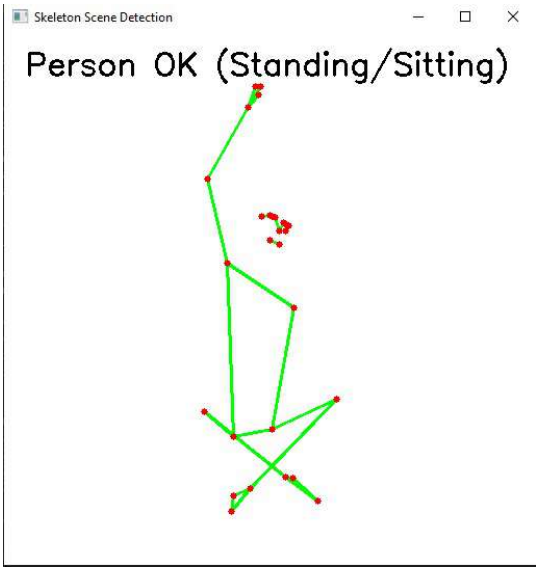
5.1. CNN Sample Result

For the CNN model the real time picture of a person sitting is being testing which was the part of the training set and it showing the correct result however its accuracy can be improved through increasing the count of training dataset.

Figure 2. Example figure **PFGPlots**.



(a) Example left figure.



(b) Example right figure.

5.2. Media Pipe Result

shows an example of a two-scenarios in which the media pipe is showing its output in the first figure it is showing the outline whereas in second is showing the extracted points from the sample picture.

This illustrates the blue print of working of under hood of media pipe for deciding the picture OK, NOT OK or NOT IN RANGE results. MediaPipe's architecture includes graph-based processing with GPU acceleration, allowing smooth inference even on devices with limited resources.

References

[1] C. Lugaresi et al., *MediaPipe: A Framework for Building Perception Pipelines*, arXiv preprint arXiv:1906.08172, 2019.

[2] TensorFlow Documentation, *TensorFlow API Reference*, 2024. Available: https://www.tensorflow.org/api_docs

[3] Keras Documentation, *Keras API Reference*, 2024. Available: https://keras.io/api/

[4] OpenCV Documentation, *OpenCV Library Reference*, 2024. Available: https://docs.opencv.org