OpenPosture Technical Report

By: Ally Ryan, Mike Nweke, Parisha Rathod

OpenPosture aims to improve the posture of seated individuals so that back, neck, and hip pain caused by poor seated posture and/or long-term and frequent sitting can be mitigated. This project utilizes OpenPose to detect key points on the body and requires two main dependencies to run: Keras and TensorFlow. OpenPosture was inspired by an MIT developed initiative titled “Seated-Posture-Recognition.” “Seated-Posture-Recognition” utilized Keras and TensorFlow to run an OpenPose model that detected the position of an individual’s back as leaning forward, reclining, or straight, the position of an individual’s hands as crossed versus uncrossed, and the position of the persons feet as kneeling versus not kneeling. Utilizing a similar OpenPose model enabled by Keras and TensorFlow, OpenPosture aims to detect back position, neck position, hand position, leg position, and if the individual’s feet are on the floor versus the ground. This paper will explore the techniques that enables OpenPose to detect posture, the key roles that Keras and TernsorFlow play, and the new code that the OpenPosture developers must tackle to further improve seated posture.

OpenPose is a computer vision system that detects and tracks human body keypoints, such as joints and limbs, from images and videos. It works by analyzing pixel data to identify key body parts and their spatial relationships. Keras is a deep learning library written in Python. It provides a simple interface for building and training neural networks, supports both convolutional and recurrent networks, and runs on both CPU and GPU. TensorFlow is an open-source machine learning framework that facilitates the creation, training, and deployment of deep learning models. TensorFlow uses data flow graphs to represent computations and provides support for various neural network architectures. The Seated Posture Recognition project utilizes the following files to achieve accurate seated posture detection through images and real time video: config\_reader, model, posture\_image, and posture\_realtime.

The config\_reader uses the ‘ConfigObj’ library to read configuration parameters from a file named 'config'. It retrieves specific parameters, including:

* **Model ID:** Specifies the specific model.
* **Box Size:** The width and height of a box drawn around a particular part of an image. It shows the area where calculations or analysis happens and outlines the space in the image being looked at.
* **Stride:** Controls how far the bounding box moves across the image during processing and decides the distance between each step horizontally and vertically. A bigger stride skips more pixels, making processing quicker with fewer calculations, while a smaller stride gives a more detailed analysis but takes longer and needs more computing power.
* **Pad Value:** The number used to fill areas outside an image when the bounding box extends beyond its edges. It ensures the bounding box's size remains consistent during computations, even if it goes beyond the image.

Once the config\_reader program specifies each parameter, which includes the aforementioned Box Size, Stride, and Pad Value, the code converts each parameter to the appropriate data type and initiates a function named ‘config\_reader()’ that reads and processes the model configuration through a dictionary containing the aforementioned parameters. The parameters specified within the model configuration established within config\_reader are essential for OpenPose to detect keypoints.

Once the configurations are set up that enables the identification of keypoints, the code contained within model is initiated to implement architectures and a deep learning model for posture detection using Keras. The model code uses VGG blocks, which are groups of convolutional layers followed by max pooling layers that are used to extract key features from data. Also established within the model code are convolutional and pooling layers, ReLU activation functions that introduce complexity to the model, concatenation that enables features to combine from different layers, and weight decay that prevents the model from memorizing the training data. Utilizing the aforementioned neural network components, the model predicts Part Affinity Fields (PAFs) and confidence maps through branches and a Lambda layer for the purpose of showing how likely it is that the captured body parts are connected and how confident the model is about each body part's location. Branching is the process of dividing the prediction task into different parts or components, thus in the Seated Posture Recognition initiative, one branch focuses on predicting the locations of key body joints while another branch predicts the connections between the joints in the image. The Lambda layer applies a math operation to each pixel in the input image to ensure consistent formatting with the data that helps the model reduce input variations and increases training speed for better model performance.

Once the configuration and model architectures are set up, images can be included within the posture\_image or posture\_realtime code to detect posture. To do this, the posture\_image/posture\_realtime code first calls the ‘process’ function to take an input image, resize it, and run it through the OpenPose model to generate heatmaps and Part Affinity Fields (PAFs), which represent the likelihood of body part locations and the association between body parts. Next, functions are run to calculate angles between body parts, check if hands are folded or not, detect kneeling posture, and visualize the detected keypoints on the image. Once the aforementioned calculations are made, the image is processed and the results are displayed to indicate whether the person was standing straight, hunchbacked, or reclined, had hands folded, or was kneeling.

The code files within the Seated Posture Recognition module can be extended to detect additional body postures such as crossed legs, feet on the floor, and neck position. To enable the detection of crossed legs, the code can be modified to analyze the relative positions of the detected key points corresponding to the hips, knees, and ankles. By comparing the coordinates of these points, it can be determined if the legs are in a crossed position through predefined thresholds or geometric relationships. For example, if the distance between the knees exceeds a certain threshold or if the angles formed by the knees and hips meet specific geometric conditions, it can be inferred that the legs are crossed. Further, the OpenPosture developers will plan to detect if an individual’s feet are on the floor by analyzing the positions of ankle key points relative to the floor plane. To further specify, if the vertical position of the ankle key points is below a certain threshold, it will indicate that the feet are touching the floor, and thus that the feet posture element is optimal. Lastly, determining neck posture will be done by analyzing the coordinates of the key points corresponding to the shoulders and neck. Specifically, functions will be created that calculate the angle formed between shoulder and neck key points relative to the vertical axis. In this case, a big angle will indicate an upright neck position while a small angle will indicate a downward neck position. By utilizing Keras and TensorFlow to initiate an OpenPose model, the OpenPosture developers will improve seated posture by incorporating neck position, feet on floor status, and legs crossed status.