OpenPosture Final Report

By: Ally Ryan, Michael Nweke, Parisha Rathod

Introduction

Poor sitting posture is known to contribute to various health issues, including lower back and neck pain, musculoskeletal imbalances, and reduced flexibility. For busy students and corporate working professionals who spend a lot of time sitting at a desk, back and neck pain can also cause lower productivity levels. Back and neck pain can be improved by adhering to the following good-seated posture components: 1) Avoid crossing the knees or ankles, 2) place the ankles in front of the knees, 3) relax the shoulders, 4) sit up straight and look forward without straining the neck, and 5) keep feet flat on the floor. Before OpenPosture, busy students and corporate working professionals who suffered from back and neck pain did not have an easily accessible tool that fully assessed their posture. In this capstone project, a UI was created to enable users to input a photo of or live film their posture and receive recommendations for what they are doing right in their sitting routines and what they could improve on.

Objectives

This project minimized the risk of neck and back pain of busy students and corporate working professionals by assessing elements of seated posture, including the alignment and position of the back, feet, knees, and neck, in a model. This model provides personalized recommendations for posture improvement based upon the model's results. Using this model, a UI was created to enable users to input a photo of or a live film of them sitting at their desk and output their postural results and recommendations.

Scope

This project focused on assessing seated posture by detecting key body points using a trained Keras model of OpenPose. This project was inspired by Seated Posture Recognition source code on GitHub that determined back position (straight, hunchback, reclined), hand position (folded vs not folded), and kneeling (feet curled behind the knees). The OpenPosture developers improved upon this source code by adding additional posture components such as neck position, feet placement, and the detection of crossed legs.

Data Sources

This project utilized a trained Keras model of OpenPose to detect keypoints of the human body. The Seated Posture Recognition source code that inspired this project tested their data using a self-made image dataset of one person sitting at a desk. The project owners took 13 pictures, which included the following: hunchback, hunchback flip, hunchback hands folded, hunchback hands folded flip, kneeling, kneeling flip, recline, recline flip, recline hands folded, recline hands folded flip, straight flip, straight hands folded, and straight hands folded flip. The OpenPosture developers recreate the aforementioned images and also accounted for the following posture components in additional images: feet on the ground, the crossing of legs, and neck position. This resulted in 144 total images for OpenPosture.

Methodologies and Model Technical Description

OpenPosture utilized Vue.JS, a JavaScript framework for building user interfaces for web applications, for frontend methodology. For a backend database, OpenPosture utilized Firebase for a real-time database to store the UI data. To create a model that accurately analyzes posture, OpenPose and OpenCV was leveraged to detect and track

keypoints of the human body from images, including for the head, shoulders, elbows, wrists, hips, knees, and ankles. This project added neck position, feet placement, the detection of crossed legs, and recommendations to the Seated Posture Recognition source code. This was done by adding new code to the model and posture_image source code and creating a new test dataset to account for the images taken by the source code's authors and OpenPosture's postural additions. Results were tested by viewing the model's output and comparing it to the actual posture in the image. Refer to the Results section for further results methodology.

OpenCV 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 utilized 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 OpenCV 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 OpenCV 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 were 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 was 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, the OpenPosture developers were able to determine if the legs were in a crossed position through predefined thresholds or geometric relationships. For example, if the distance between the knees exceeded a certain threshold or if the angles formed by the knees and hips met specific geometric conditions, it was inferred that the legs were crossed. Further, the OpenPosture developers detected if an individual's feet were 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 were below a certain threshold, it indicated that the feet were touching the floor, and thus that the feet posture element is optimal. Lastly, determining neck posture was done by analyzing the coordinates of the key points corresponding to the shoulders and neck. Specifically, functions were created that calculated the angle formed between shoulder and neck key points relative to the vertical axis. In this case, a big angle indicated an upright neck position while a small angle indicated a downward neck position.

Results

To capture results for the OpenPosture model, the developers sampled 20 photos and compared the predicted results to the actual results for kneeling, back

position, neck position, and hand position static photos. The developers compared each of the postures to each other to determine which one performed best and why. The metrics captured are confusion matrix related data points, included True Positive, True Negative, False Positive, False Negative, Recall, Precision, Sensitivity, and Specificity. In this model, positive values represent optimal postures.

The below confusion matrix for kneeling, **Image 1**, offers an overview of the classification performance in distinguishing between two postural states: "Kneeling" and "Not Kneeling," where "Kneeling" represents a seated posture where the toes are positioned behind the knees, and "Not Kneeling" signifies an optimal posture. The matrix reveals that out of the instances classified as "Kneeling", the model correctly identified 2 cases (True Negative), while incorrectly classifying 0 as "Not Kneeling" (False Positive). Conversely, among the instances labeled as "Not Kneeling", the model correctly identified 16 cases (True Positive) but misclassified 2 as "Kneeling" (False Negative). As such, the model correctly classified 90% of the sample, or 18/20 kneeling instances. Refer to **Image 4** for precision, recall, specificity, and sensitivity for kneeling. Reasons for the inaccurate values are because the position of the knees versus the toes are midway between kneeling and not kneeling.

Kneeling Confusion Matrix		
	Kneeling	Not Kneeling
Kneeling	2	0
Not Kneeling	2	16

Image 1

The below confusion matrix for back position, **Image 2**, offers an overview of the classification performance in distinguishing between two postural states: "Hunched/Reclined" and "Straight," where "Hunched/Reclined" represents a seated

posture where the back is angled forward or backwards, and "Straight" signifies an optimal posture. The matrix reveals that out of the instances classified as "Hunched/Reclined", the model correctly identified 12 cases (True Negative), while incorrectly classifying 0 as "Straight" (False Positive). Conversely, among the instances labeled as "Straight", the model correctly identified 8 cases (True Positive),and misclassified 0 as "Hunched/Reclined" (False Negative). As such, the model correctly classified 100% of the sample, or 20/20 back position instances. Refer to **Image 5** for precision, recall, specificity, and sensitivity for back position. Reasons for all accurate classifications is because the images sampled clearly capture straight, reclined, and hunched postures.

Back Confusion Matrix		
	Hunched/Reclined	Straight
Hunched/Reclined	12	0
Straight	0	8

Image 2

The below confusion matrix for hand position, **Image 3**, offers an overview of the classification performance in distinguishing between two postural states: "Folded" and "Not Folded," where "Folded" represents a seated posture where the hands are crossed and potentially misaligning the spine, and "Not Folded" signifies an optimal posture. The matrix reveals that out of the instances classified as "Folded", the model correctly identified 3 cases (True Negative), while incorrectly classifying 0 as "Not Folded" (False Positive). Conversely, among the instances labeled as "Not Folded", the model correctly identified 14 cases (True Positive), and misclassified 3 as "Folded" (False Negative). As such, the model correctly classified 85% of the sample, or 17/20 hand position

instances. Refer to **Image 5** for precision, recall, specificity, and sensitivity for hand position. Reasons for inaccurate classifications is because the images sampled captured arms that were parallel and thus indistinguishable.

Hands Confusion Matrix		
	Folded	Not Folded
Folded	3	0
Not Folded	3	14

Image 3

The below confusion matrix for neck position, **Image 4**, offers an overview of the classification performance in distinguishing between two postural states: "Forward/Backward" and "Straight," where "Forward/Backward" represents a seated posture where the neck is angled downwards or upwards and potentially causing neck pain crossed, and "Straight" signifies an optimal posture. The matrix reveals that out of the instances classified as "Forward/Backward", the model correctly identified 4 cases (True Negative), while incorrectly classifying 2 as "Straight (False Positive). Conversely, among the instances labeled as "Straight", the model correctly identified 14 cases (True Positive) and misclassified 0 as "Forward/Backward" (False Negative). As such, the model correctly classified 90% of the sample, or 18/20 neck position instances. Refer to **Image 4** for precision, recall, specificity, and sensitivity for neck position. Reasons for inaccurate classifications are because the position of the neck was in between straight vs forward or backward and thus not clearly distinguishable.

Neck Confusion Matrix		
	Forward/Backward	Straight
Forward/Backward	4	2
Straight	0	14

Image 4

As shown in **Image 5**, the metrics vary across different posture elements due to differences in classification accuracy and the ability of the model to correctly identify instances of each posture. In terms of precision, "Kneeling", "Back", and "Hands" all achieved a perfect score of 100%, indicating that when the model predicts these postures, it is highly accurate. "Back" had the highest recall, specificity, sensitivity, and percent correct, suggesting that it performed optimally in terms of both identifying true positive and true negative instances, resulting in a flawless classification. "Hands" had a slightly lower recall and sensitivity compared to "Kneeling" and "Back", indicating that it might have missed identifying some true positive instances. However, it still maintained a high precision, resulting in a relatively high percent correct. "Neck" had a relatively high precision but lower specificity, suggesting that while it correctly identified many positive instances, it also misclassified some negative instances. However, its overall classification accuracy, as indicated by percent correct, was still quite high. Overall, based on the provided metrics, "Back" appears to be the most optimal posture element as it achieved perfect scores across all metrics, indicating flawless classification performance.

Posture	n	Precision	Recall	Specificity	Sensitivity	Percent Correct	Percent Incorrect
Kneeling	20	1	0.89	1	0.89	90%	10%
Back	20	1	1	1	1	100%	0%
Hands	20	1	0.82	1	0.82	85%	15%
Neck	20	0.88	1	0.67	1	90%	10%

Image 5

Precision measures the accuracy of positive predictions. In this model, high precision means that when the model predicts a posture as optimal, it is highly likely to be correct. High precision ensures that the model minimizes false positives, meaning it is less likely to incorrectly classify non-optimal postures as optimal and prevents the provision of incorrect guidance or feedback to the user. Kneeling, back, and hands have a higher precision than neck, indicating a more favorable performance for kneeling, back, and hands.

Recall (Sensitivity) measures the ability of the model to correctly identify all optimal posture instances. High recall indicates that the model effectively captures all instances of the optimal posture from the dataset. High recall ensures that the model does not miss identifying any instances of the desired posture, minimizing the risk of failing to provide feedback or guidance when the user is in the optimal position. Back and neck performed the best, kneeling performed third best, and hands performed third best.

Specificity measures the ability of the model to correctly identify all non-optimal instances. In this model, high specificity means that the model effectively distinguishes non-optimal postures from the optimal one. This ensures that when the user is not in the optimal posture, the model correctly identifies this condition, preventing false alarms or incorrect feedback. All posture elements performed perfectly except for the neck, indicating that the model may suggest incorrect recommendations for neck posture. In this model, specificity is likely the most important accuracy measure because it enables users to be correctly provided feedback for non-optimal postures.

The OpenPosture developers originally intended to compare the performance metrics from the static photos to the performance metrics of the live video recording. However, the developers struggled to adequately capture metrics recorded over a period, as the model continuously captures data. The OpenPosture developers will leave the performance metrics for live results for future work. Despite not capturing the results for live results, the OpenPosture developers recognized that the results are very similar to the high accuracy of the static photos because the model is the same. In the future, the OpenPosture developers also intend to fine tune the model so that it detects parallel hands and in between neck and kneeling positions.

Future Work

The following are areas where the OpenPosture developers see potential for future work. The objectives would help the OpenPosture device, once manufactured to be more user friendly and reach a wider audience.

1. **Object Detection:** Integrating real-time object detection capabilities within the OpenPosture system would enhance its ability to recognize and provide insights on specific objects in the user's environment. Such insights might include lowering the chair, raising the desk, or moving the keyboard forward to enable better posture and mitigate pain.
2. **Single-Device Use:** Optimizing OpenPosture for single-device use could eliminate logistical challenges for users with a single device and streamline the user experience by removing the need for additional hardware.

3. **Camera Positioning Guidance:** Incorporating visual aids, instructions, or alarms/audio cues within the app to assist users in positioning the camera effectively would improve posture assessment accuracy.
4. **Customization:** Offering customization options within the app regarding workout capabilities, known medical diagnoses (e.g., scoliosis), age, or physical characteristics (e.g., amputated body part) would cater to diverse user preferences and needs.
5. **Subscription Management:** Establishing and allowing options for different subscription plans would allow users to choose the version of the app that best suits their needs and financial interests.

Conclusion

In conclusion, the OpenPosture project successfully improved upon the Seated Posture Recognition Source Code developed by the MIT students by adding code for the feet and position of the neck, as well as workout recommendations. If going to market, OpenPosture would be a successful and highly profitable product, as it would be marketed as a subscription package to Big4 Accounting firms. As a result of OpenPosture, the back and neck pain of busy students and corporate working professionals is easier to mitigate than ever before.

Additional Project Info

Team members Ally Ryan, Michael Nweke, and Parisha Rathod all contributed equally to this project. Notable tasks for each member include:

- **Ally Ryan:** project management, main deliverable creator (poster, summary and technical reports), workout recommendation researcher, business

venture video and powerpoint development, results code development and output

- **Parisha Rathod:** Neck code developer, researchathon attendee, tested data for accuracy, created final poster, assisted with deliverables (summary and technical reports and powerpoints)
- **Michael Nweke:** Main backend and frontend developer, created demo videos, tested data for accuracy