PosturePal: Real-Time Posture Classification with a Laptop Webcam

Ethan Weber
MIT EECS
Cambridge, MA
ejweber@mit.edu

Moin Nadeem
MIT EECS
Cambridge, MA
mnadeem@mit.edu

***Abstract*—Bad posture is a problem that affects many people which can lead to arthritis among many other health conditions. We attempt to alleviate this issue with PosturePal, an application that classifies user posture from the front-facing camera on any laptop. By utilizing various computer vision and machine learning techniques, we go from 2D RGB images of individuals to binary classification of posture (good or bad). After classifying posture, we can then notify the user when they are sitting with bad posture in real-time. We present our work in this paper, and we explain the future goals and implications of such a system.**

***Index Terms*—Computer Vision, Posture, Classification, Human Keypoints, Machine Learning**

I. Introduction

Most people–especially students–sit in front of their laptops all day. Furthermore, it's estimated that 80% of people will experience back pain at some time during their lives [1]. Given the recent success in computer vision algorithms, we believe that we can address this issue with laptop webcams. Currently people use their laptop webcams for FaceTime, Skype, etc., but few applications are using them for health benefits! For this reason, we propose PosturePal: the project that monitors your posture with your webcam and alerts you when sitting improperly.

Inspired by OpenPose [2]–a convolutional neural network that can detect keypoints of people from 2D images, PosturePal uses various classification algorithms on the keypoints to determine whether someone's posture is good or bad. We have created an extensive pipeline to perform this analysis and classification, which we describe in this paper. At a high level, we are trying to get PosturePal to go from an RGB image to keypoints to classification. We show an example of what we are performing classification on in Figure 1.

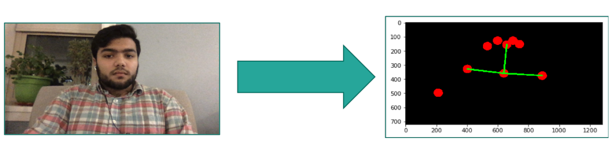

Fig. 1. We convert the RGB image to keypoints to perform classification on. (This step is done with OpenPose.) Here we show what those points look like. The points connected in green are those that we believe hold the most relevant information. We performed experiments where we compared classifying just these points vs. all the keypoints, and just these points had the best results.

II. Related Work

To the best of our knowledge, there is minimal work in the area of classifying posture from 2D images. We did, however, find one product on the market called Posture Monitor [3], which utilizes an Intel RealSense 3D camera to detect posture based on depth data while sitting in front of a computer. Furthermore, this product only works with the Windows OS. PosturePal, on the other hand, will work for anyone with a standard laptop (assuming it has a webcam).

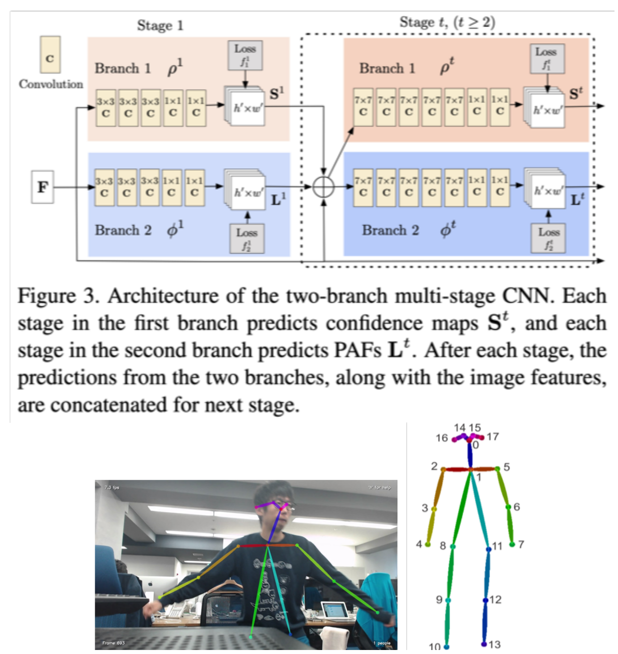

Fig. 2. Here we show a figure from the OpenPose paper depicting its network architecture. It works by predicting confidence maps for both keypoints and the segments that connect the keypoints. A matching algorithm is then run to produce the connected keypoint images (illustrated by the connected keypoints in the bottom left image). The bottom right image shows which keypoints are predicted by the network. In particular, there are 18 keypoints that are predicted for; they are indexed according to the numbers shown here.

We are quite confident our work is novel, but we couldn't have done it without using work from previous research. In particular, we rely heavily on OpenPose, a network that takes RGB images and outputs 2D keypoints of humans. We show a diagram from the OpenPose paper in Figure 2. OpenPose outputs predicted joint locations for all humans in an image, but we make the assumption that only one user will be using their computer. Therefore, we only take the first human returned from the network.

In additional to OpenPose, there have been other papers that try to solve the human keypoint detection problem, such as DensePose [4]. DensePose is a network that goes from an RGB image of people to reconstructing the 3D surface of the human bodies in the scene. In relation to our work, this would be equivalent to an OpenPose network that predicts for many more than 18 joint locations. However, we did not explore DensePose due to limited time and the hope that 18 keypoints is sufficient for classification (although only the top half of keypoints are visible from a laptop webcam point of view anyways).

III. Pipeline and Data Setup

In this section, we go over our pipeline and explain how we convert images into a vector classification problem. We begin with the software stack and explain the steps leading up to classification algorithms.

A. Software Stack

In Figure 3, we show our software stack pipeline. We break the software stack into 3 main components: **server side (blue)**, **data management (green)**, and **classification analysis (red)**. The diagram doesn't illustrate complexity of our code, but it effectively conveys the high level components of our pipeline. Now we explain the sections in more detail.

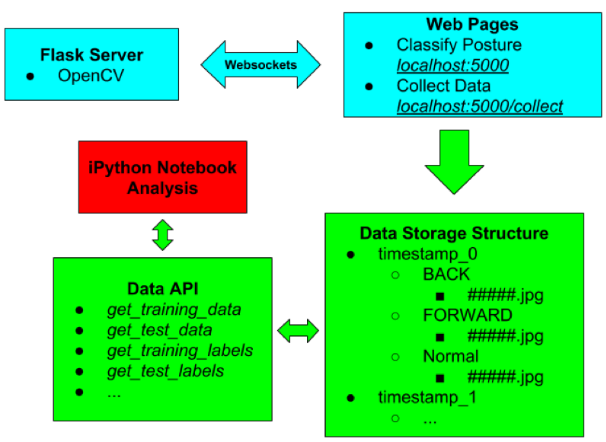

Fig. 3. This is a diagram depicting the software stack we created for this project. Blue illustrates the server side, green is for data management, and red is for classification analysis.

1) Server Side: We use a Python3 Flask [5] server that runs locally on someone's machine. We chose to use a local server to reduce privacy concerns. Our application can be run entirely offline on the user's computer. There is a chance that we do everything in the browser in the future, but for now we are trying to minimize privacy concerns.

The Flask server utilizes the OpenCV Library [6] to handle reading images from the webcam. We use websockets to handle the communication between the server and the web frontend. Websockets help speed up network transmission times by removing HTTP headers, and therefore allow for drawing webcam images in the browser in real-time. We transmit the images using Base64 image encoding and decoding on the server and client (browser) side respectively.

Furthermore, we currently have two main webpages in our frontend code. The first is a site for real-time posture classification. Two screenshots from this webpage are shown in Figure 4. These images show a version of our application where clicking the "Run Inference" button will classify the image based on the classification algorithm and parameters we have set at the time.

The other very important webpage we have is for streamlining data collection. Figure 5 illustrates this process. A user can sit in front of their computer with this page open, click the "Start Collection" button, and then replicate the postures shown in the figure. Interactive highlighting and text instructions show the user how to collect data correctly. Our server will then take the images collected and save them to as a timestamped sequence. All the image sequences are saved to folders of the following structure:

- timestamp
  - BACK (folder of images)
  - FORWARD (folder of images)
  - NORMAL (folder of images)

In our work, we consider both BACK and FORWARD images to be bad posture and NORMAL images to be good posture. Figure 5 shows what we mean by BACK, FORWARD, and NORMAL posture. For most sequences that we selected, we took 20 images of each posture type. Therefore, this amounts to 60 images for a given collection trial, where 40 are bad and 20 are good postures.

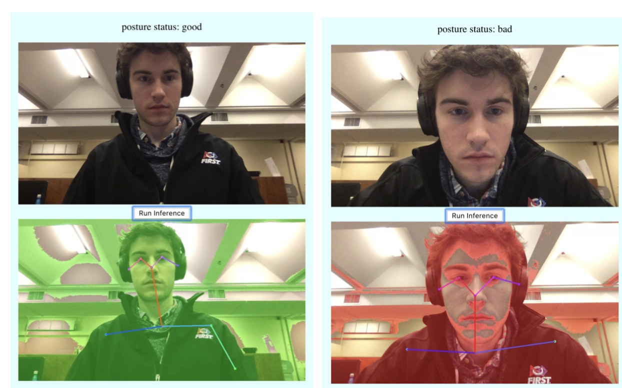

Fig. 4. Here we show two screenshots from our real-time posture classification page. This page allows us to run whichever posture classification algorithm we are using at the time.

2) Data Management: After creating code that could collect images and store them as organized sequences, we wrote a data API. The data API's purpose is to convert the RGB images to vectors. The preprocessing involves many steps, so we explain those here.

First of all, we utilize an implementation of OpenPose from GitHub [7]. We use the same neural network weights as those in the original OpenPose paper from CMU [2]. Using this

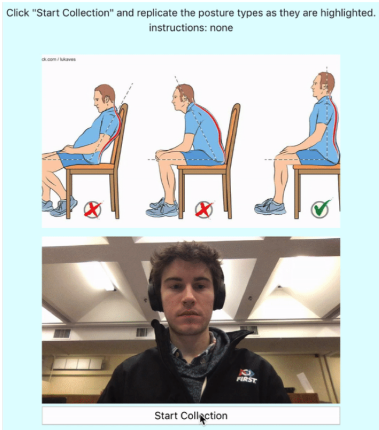

Fig. 5. This is the webpage that we use to collect data. We had 4 different people follow the steps (BACK, FORWARD, NORMAL posture left to right) outlined in the diagram. The images from each trial are saved as a timestamped sequences in labeled folders.

code, we can run inference on an RGB image and get the keypoints output in normalized coordinates (values ranging from 0 to 1 instead of absolute pixel indices). After running inference for an image, we have 18 keypoints to work with according to the diagram in Figure 2. We then create a 36 dimensional vector by combining the x and y values for each of the 18 keypoints into a single vector in the following way:

$$[x_0, x_1, ..., x_{16}, x_{17}, y_0, y_1, ..., y_{16}, y_{17}]$$

Note the x and y values for missing keypoints are 0. We then postprocess these intermediate vectors by subtracting all non-zero keypoints by the position of the keypoint that corresponds to the chest. The chest keypoint has index 1, as shown in Figure 2. This helps ensure that our vectors will work regardless of any displacement of the user. Quite simply, we make all points in the vector relative to the chest keypoint location.

After taking the images and converting them to a vectors in this way, we then create an organized dictionary data structure for every sequence that allows one to quickly grab a list of vectors for all posture types (BACK, FORWARD, and NORMAL). Finally, we serialize the dictionary and save it as a dictionary with the same timestamp as the sequence. We do this by saving the sequences as pickle files in Python3.

With all the sequences processed to vectors and saved to pickle files, we now can load vectors very quickly. Given this, we wrote some functions that allow us to query for a training and test set from all of the processed sequences. By providing this layer of abstraction, we quickly reduced our problem to vector classification. We now move onto processing the keypoints, which is arguably the fun part!

3) Classification Analysis: With the Data Management taken care of, we use iPython Notebooks to experiment with data processing and visualizations. The data API allows us to specify a portion of our data to be for training and a portion for testing. We outline our findings in the **Methods and Experimental Results** section.

IV. Methods and Experimental Results

Here we explain the process we went through for classifying keypoints as good and bad posture. Before we dive into the details of what worked best, we first explain our initial experiments to get some intuition for the problem.

A. Feasibility Test

We started our work with a feasibility test by running PCA (principal component analysis) on the keypoints in our collected data set. PCA works by finding the linear projection that maps a vector into a lower dimension while maximizing the variance along each of the new dimensions. Here we project the 36-dimensional vectors to 2 dimensions.

We show results of running PCA for individual sequences in each of the 7 small plots. The large plot is of the 7 sequences (which has in total 4 different people). Upon inspection, it's clear that some of the small plots look better than others. For instance, the bottom left plot appears to be very separable (meaning the different types of postures are plotted far away from each other). However, the bottom right plot is not very separable. We believe this is due to ”keypoint dropout”, which means that sometimes not all important keypoints are visible. For example, sometimes a shoulder will not be detected in an image. This introduces high variance, which we explain later with our data augmentation experiments.

Looking at the large plot (with all sequences), we note that the projection in 2 dimensions is not clearly separable. However, we were quite confident that given the few good examples of individual sequences we could still perform effective classification. In particular, we had the intuition that performing (potentially nonlinear) classification on the vectors **not** projected to 2 dimensions would have good results. After the feasibility test, we continued on with vector classification work.

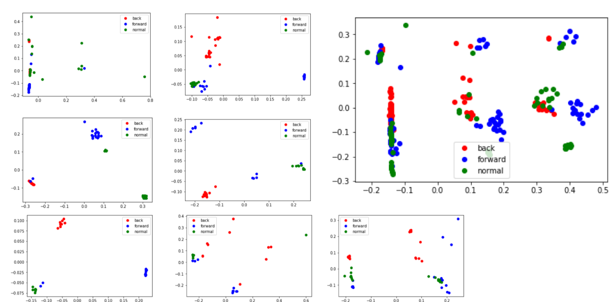

Fig. 6. Here we show results of running principal component analysis (PCA) to project the 36 dimensional vectors into 2 dimensional vectors. The small plots are of individual sequences, while the the large plot is on all the data (4 people and the 7 individual sequences).

B. Data Extrapolation

Constrained by limited resources and realizing that our data set was small, we wanted to experiment with generating more data from our relatively small dataset. We did this by taking our full dataset and estimating Gaussian distributions for each of the keypoints in 2D space. Recall that to create a vector, we go from an RGB image to keypoints (Figure 1). We can then take all of these keypoints from the dataset, find their mean and covariance, and then create plots like those shown in Figure 7 for each of BACK, FORWARD, and NORMAL posture positions. Each color illustrates the the 2D Gaussian distribution of one of the joints.

Here we show some psuedocode for estimating the Gaussian distributions for each of the keypoints.

```
import numpy as np

# dictionary for the distributions
dist = {}

for posture_type, vectors in dataset:
    # holds means and covariances
    means = []
    covs = []
    for i in range(18):
        points = []
        for vec in vectors:
            # this gets the (x, y)
            # value at keypoint index i
            point = (vec[i], vec[i+18])
            points.append(point)
        points = np.array(points)
        mean = np.mean(points)
        cov = np.cov(points.T)
        means.append(mean)
        covs.append(cov)

    dist[posture_type]["means"] =\
        np.array(means)
    dist[posture_type]["covs"] =\
        np.array(covs)
```

After obtaining all the distributions for each keypoint, and for each type of posture (BACK, FORWARD, and NORMAL), we can then sample from the distributions to get an arbitrary number of vectors to perform classification on. By using this method on the full dataset and generating a sufficient number of vectors, we are effectively forcing the classifiers to be more robust to noise. We evaluate the robustness of this technique for some experiments in the **Classifiers** subsection.

In some of the plots, there are keypoint distributions with large variances in the direction that intersect with the center origin (0,0). This is due to the ”keypoint dropout” problem. When there are keypoints not detected with OpenPose, they take the value (0,0). This introduces high variance and a lot of noise. After some experiments, we decided to ignore all vectors with ”keypoint dropout” on the chest, nose, and shoulder keypoints (the 4 points connected in green in Figure [**?**]).

C. Classifiers

Once we had validated the initial idea, we decided to evaluate our results on several different models.

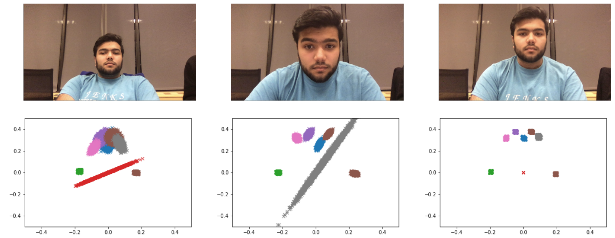

Fig. 7. Here we show the plots of representing each keypoint by a Gaussian distribution for BACK, FORWARD, and NORMAL postures (left to right). Each color represents a keypoint.

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
Gradient Boosted Trees	*0.91*	*0.92*	*0.90*	*0.91*
Random Forest	0.86	0.87	0.86	0.86
Support Vector Machines	0.82	0.83	0.83	0.80
Logistic Regression	0.82	0.83	0.83	0.83
Gaussian Naive Bayes	0.66	0.78	0.66	0.54

TABLE I
ABILITY TO MODEL A BINARY CLASSIFICATION TASK ON AUGMENTED DATA.

After finishing building our dataset, we had 339 images uniformly distributed across all three classes before any augmentation had occurred. We frame our classification problem as two different tasks: one problem formulation where we group BACK and FORWARD postures as *bad* postures, and a NORMAL posture as *good*. We also provide another formulation as a multi-class classification task where we simply attempt to classify each image into the appropriate category for a basis of comparison.

We attempted to perform classification on these tasks with the following five different classifiers:

1) Gradient Boosted Trees
2) Random Forest Classifiers
3) Support Vector Machines
4) Logistic Regression
5) Gaussian Naive Bayes

Our implementations of the classifiers came from *sklearn*, using the default hyperparameters for each model. Lastly, we also explore the performance of our classifiers with respect to our data augmentation methods as a means of determining how robust our data augmentation strategies may be. Furthermore, we performed many experiments with the 36-dimensional vectors but only report results for the 8-dimensional vectors (the points connected in green in Figure 1) because they were much better.

1) Binary Classification: Table I demonstrates our results on augmented data without adding any hand-crafted features. In order to ensure that our results were not overfitting on a random test set, we used 10-fold cross validation.

It is clear to see that Gradient Boosted Trees performed the best on our dataset, followed by a Random Forest Classifier. In particular, we wanted to highlight two aspects of our binary classification results that provided insights about our data.

First, a Naive Bayes approach performs quite poorly on our dataset. We believe that this is due to the fact that the independence assumption in Naive Bayes doesn’t hold: each keypoint is not conditionally independent of another. For example, given the fact that the bilateral symmetry in

Classifier	[Front, Back, Normal] Accuracy	Precision	Recall	F1	[Front, Normal] Accuracy	F1	[Back, Normal] Accuracy	F1	[Front, Back] Accuracy	F1
Gradient Boosted Tree	0.88	0.89	0.88	0.88	0.94	0.94	0.88	0.88	0.96	0.96
Random Forest	0.85	0.87	0.86	0.86	0.91	0.91	0.88	0.88	0.92	0.93
Support Vector Classifier	0.81	0.87	0.81	0.82	0.86	0.87	0.86	0.85	0.98	0.99
Logistic Regression	0.79	0.87	0.79	0.79	0.86	0.87	0.83	0.82	0.99	0.99
GaussianNB	0.75	0.80	0.75	0.75	0.81	0.80	0.82	0.82	0.96	0.96

TABLE II

TABLE II DEMONSTRATES THE RESULTS OF AN ABLATION STUDY ON THE INPUT DATA. IN PARTICULAR, THIS DEMONSTRATES THE LARGE DIFFERENCE BETWEEN THE FRONT AND BACK CLASS, WHILE DIFFERENTIATING NORMAL FROM EITHER NEGATIVE CLASS IS SLIGHTLY MORE DIFFICULT.

Classifier	*Without Augmentation* **Accuracy**	**F1**	*With Augmentation* **Accuracy**	**F1**
Gradient Boosted Tree	0.88	0.88	0.82	0.82
Random Forest Classifier	0.87	0.87	0.80	0.81
Support Vector Classifier	0.81	0.82	0.72	0.72
Logistic Regression	0.79	0.79	0.72	0.73
Gaussian Naive Bayes	0.75	0.75	0.84	0.84

TABLE III

TABLE III DEMONSTRATES THE IMPACT OF OUR AUGMENTATION STRATEGY UPON OUR CLASSIFIERS.

the human body, and the fact that the keypoints are centered around the chest, it is clear to see that the left shoulder is conditionally dependent upon the right shoulder. As a result, we were able to rule out this class of models from future consideration.

Second, we notice that using Logistic Regression or a Support Vector Machine has produced equivalent classification results. At first, we had a hard time understanding the reasons as to why, since they both are quite different classification methods. However, given the fact that we used a linear kernel for the Support Vector Machine, the only difference in the two methods is a hinge loss versus logistic loss, which are nearly equivalent for small inputs.

2) Multiclass Classification: We extend our insights in the previous section to a multiclass problem formulation, as well as perform an ablation study on our classifier to understand its strengths and weaknesses.

Our results show that there is very little difference between a multiclass classification task on all three categories and a binary classification task. Therefore, it should be best to treat this as a binary classification task for any application of the work.

However, these studies also indicate that classifying between a FRONT and BACK posture, and a FRONT and NORMAL posture have similar results, both of which perform better than a BACK and NORMAL task. This implies that our features between BACK and NORMAL may lack sufficient distinguishing features, and we believe this is an area for future work. One particular intuition is that the main difference is that NORMAL and BACK postures only differ by depth rather than a strong difference in keypoints, suggesting the need for depth perception as a feature.

3) Data Augmentation: Since we have formulated a custom data augmentation strategy, we also believe that we should evaluate the effectiveness of this strategy on our classifiers. We perform an ablation study on our data augmentation strategy to evaluate its effectiveness.

We see that our data augmentation strategy actually *hinders* performance of our model in all but one case. Since our augmentation strategy assumes that keypoints follow a Gaussian distribution, it makes sense that *Gaussian Naive Bayes* model would increase in performance once we add noise that follows a Gaussian distribution. Intuitively, we believe that our data augmentation strategy is reasonable, but it is possible that the underlying distribution does not follow a Gaussian distribution, or that we have parameterized our Gaussian distribution incorrectly.

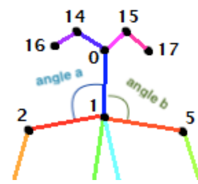

Fig. 8. Figure 8 demonstrates the intuition that the cosine angles between vectors may lead to improved performance

4) Handcrafted Features: Finally, we had an intuition that a helpful feature to hand-engineer may be the cosine angle between each pair of points. For example, the cosine angle between one's left shoulder and neck may be useful towards inferring posture. Similarly, the cosine of the angle formed by the vector from each shoulder to chest point may also provide valuable information. To convince yourself of this, consider the angles an individual's shoulders make when sitting properly (often 180-degrees), versus when leaning forward or back.

Classifier	*With Angles* **Accuracy**	**F1**	*Without Angles* **Accuracy**	**F1**
Gradient Boosted Trees	0.91	0.91	0.89	0.90
Random Forest	0.89	0.89	0.85	0.85
Support Vector Classifier	0.83	0.83	0.80	0.80
Logistic Regression	0.83	0.83	0.80	0.80
Gaussian Naive Bayes	0.66	0.54	0.66	0.54

TABLE IV

TABLE IV DEMONSTRATES THE IMPACT UPON OUR PERFORMANCE THAT OUR HAND-CRAFTED FEATURES HAVE. WE CAN SEE THIS HELPS IMPROVE PERFORMANCE IS ALL KEY AREAS.

Table IV demonstrates the impact in performance that this handcrafted feature has. We can see that this leads to an increase in performance across all key areas. This aligns with our expectation that the cosine distance should be a helpful metric for the model to learn, and leaves us to believe that developing other hand-crafted features may be a useful starting point for future work.

V. FUTURE WORK

In terms of future work, we have many plans to further develop our work. In particular, we have some limitations that we plan to address with more time. Furthermore, we plan to polish the application to make it more user-friendly. We also plan to try out some new techniques for posture detection (such as using depth information), and we conclude that other health issues could be addressed in similar ways by using laptop webcams.

A. Addressing Limitations

Currently we have some limitations to our current approach for posture classification. We explain those here and then explain how we may improve our results. One major limitation is "keypoint dropout", which we explained in the **Data Extrapolation** section. Essentially, we should not be making predictions on data that does not detect relevant keypoints. For instance, if the laptop image only detects one shoulder, we should **not** classify posture. Based on human intuition, we would not be able to classify posture based on points with a shoulder detection missing. This is why we ignore vectors with "keypoint dropout" in our experiments, but we wish to address this issue better in the future.

1) Trying to Generalize for all People: Instead of using a dataset of multiple people, it might be best to only have a calibration stage for the user of interest. In particular, we realize that not everyone may use their laptops in the same way. Some sit closer to their laptop than others, some may sit further way, etc. Our **Data Extrapolation** may be sufficient to generate enough data from just one person after some initialize calibration. This is something we will look into in the future.

B. Depth Estimation

The current posture monitoring product on the market (which we are aware of) uses depth to make the classification of good and bad posture. However, Posture Monitor makes use of an actual depth camera. We believe we could replace this need with a learned approach. In the field of self-driving cars, there has been a lot of work on using deep learning to transform a single RGB image to a depth map. We feed a webcam image through a depth prediction network [8] (trained on self-driving car data) in Figure 9. Clearly the results aren't the best because images of people in front of their laptops were not seen during training, but we are confident that retraining a similar network on our own data (that better matches the expected distribution of images we will see in front of a computer) will achieve very good results. This may be a good way to go about the posture classification task. Posture Monitor appears to be successful with the use of 3D depth data because it greatly simplify the problem of posture classification.

C. End-to-End Neural Network

Another approach to the posture classification problem would be to use an end-to-end convolutional neural network. Image classification with convolutional neural networks has shown to be very successful in general, but to get good results requires a lot of training data. For this reason, we did not use this approach for our project. Maybe in the future we will try an end-to-end neural network if we crowdsource data collection or pay people on a system such as Amazon Mechanical Turk [9] to get images to work with. Due to resource and time constraints of this project, we went with the more predictable route of keypoint classification based on the work of OpenPose.

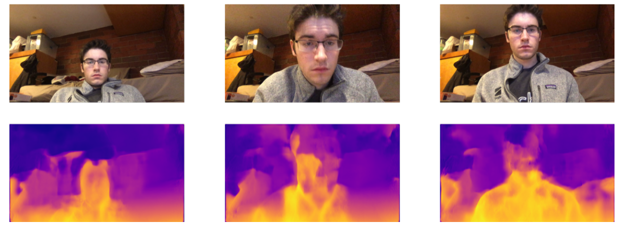

Fig. 9. Here we show an example of going from an RGB image to a predicted depth image. We are using a pretrained network [8] on the Cityscapes Dataset, so the results here are not very good for these images.

Fig. 10. An illustration depicting a possible end-to-end neural network. "PostureNet" (as depicted here) would require a large amount of training data, which is why we chose to the 2-dimensional keypoint approach based on OpenPose work.

D. Conclusion

Given the success of this project, we are very excited about the possibility of making PosturePal accessible to everyone. We hope to help people address their posture problems in a convenient manner. Furthermore, we are interested in applying similar techniques to resolve other problems such as face touching, distraction, etc. We look forward to releasing more work in this unexplored area. We have released all code and approved images to promote reproducibility of our work.

E. Contributions

Ethan worked on creating the software pipeline and integrating OpenPose into the code. He did preliminary tests with PCA and worked with Moin on the data loader, vector preprocessing, classification, and the server / client infrastructure.

Moin Nadeem worked on preprocessing keypoints into vectors, feature engineering, and classification algorithms. He worked with Ethan on writing a real-time inference engine, the Data Preprocessing / Loading API, and the server and client infrastructure.

References

[1] "Back pain facts and statistics." https://www.acatoday.org/Patients/Health-Wellness-Information/Back-Pain-Facts-and-Statistics.

[2] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[3] "Posture monitor." https://posturemonitor.org/.

[4] I. K. R{iza Alp Güler, Natalia Neverova, "Densepose: Dense human pose estimation in the wild," *arXiv*, 2018.

[5] "Flask." http://flask.pocoo.org/.

[6] "Opencv library." https://opencv.org/.

[7] "tf-pose-estimation github repository." https://github.com/ildoonet/tf-pose-estimation.

[8] "monodepth github repository." https://github.com/mrharicot/monodepth.

[9] "Amazon mechanical turk." https://www.mturk.com/.