Correct Pose Detection
12/19/22

||Won Chung, Seung Hoon Jung, Minji Kim||

A. Introduction

As a result of the introduction of computers, mobile phones, and other devices, individuals now participate in fewer leisure-time physical activities than in the past. Although physical activities have numerous benefits, such as preventing harmful weight gain, lowering feelings of anxiety, and enhancing cognitive function [1], individuals prefer to spend their time with electronic devices, which is a serious problem in the modern world. To overcome this social problem, we wanted to ask ourselves the question of "How can we create a helpful model or application that could lead individuals to find balance between the technology that people like and the physical activity that can keep them healthy?"

As a possible solution to the social issue, we have decided to design a program that analyzes a user's yoga image to evaluate if their poses are correct. Specifically, our model will analyze the image of the user's yoga poses in order to identify whether or not the user accomplished the action correctly. If we are able to establish such a program, we feel it can be expanded into an application that encourages people to exercise.

Our brief initial approach to properly construct our model was that we will train the model with about 1000 images of experts doing various yoga poses in order to capture the locations and geometrical angles of specific body joints. Then, our model will evaluate if the The application will then determine whether the geometric angles of the user's joints fall within an acceptable range. The final objective of the project is not only to identify the physical posture of the user, but also to advise them if they are following the ideal exercising posture by comparing the joint angles of the user to permitted ranges in the model.

B. Related Works

The idea of visualizing objects as Pictorial Structure (PS), such as human pose, was introduced by Fishler and Elschlager [2]. Many researchers and computer scientists have generated models based on the idea, and various PS-based models have been developed and used for many practical reasons. One of the popular PS models is MediaPipe [3], created by Google, and it mostly detects the human body with 33 main key points.

Each key point has data related to x, y, and z coordinates and visibility. The initial approach for training included only coordinates with nested-if classifiers, and there was a limitation as the coordinates of key points are highly varied by different images due to the size of the object or the existence of other objects. Vij [4] try to find a way to count the push-ups based on the angles of key points. The method reduces the range of variation among photos with the same pose, which led us to develop the first model with a mixture of coordinates and angles.

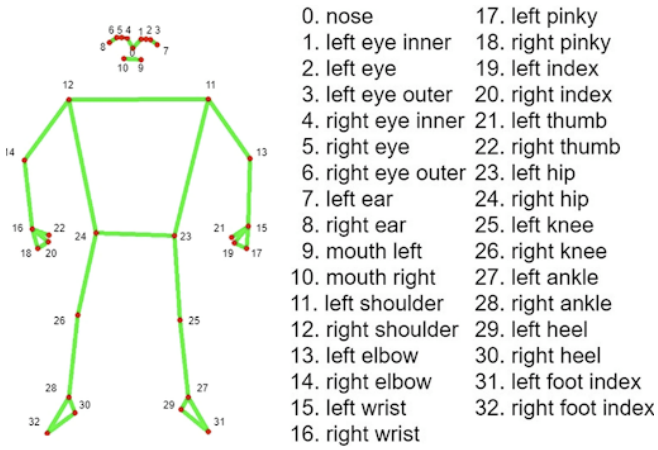


Figure 1. List of Key Points and connections between them to create a skeleton model.

Elforaici et al. develop posture recognition with different classifiers, such as SVM and KNN. They applied five classifiers (linear discriminant, quadratic discriminant, linear SVM, quadratic SVM, and cubicSVM) to apply different feature sets, such as geometrical angles, pairwise joint distances, and the combination of the 2 types of features. Our other model combines the advantages of both MediaPipe and Elforaici's approach to estimating the poses. With the data regarding the x and y coordinates of key points in MediaPipe, MLP, KNN, and logistic regression classifier are applied. Unlike Elforaici includes both 2D and 3D poses of people, our dataset focuses on real world 2D images only.

C. Method and Experiment

Our initial attempt was to develop a MediaPipe Pose Detector that could detect a person's yoga pose and create a pose detection skeleton model that can track the body's torso, limbs and facial expressions in order to classify a yoga pose out of 5 different poses. Instead of using a pre-trained model, we decided to propose a model ourselves so that we can train it with our own dataset and test the accuracy of the model.

C.1. Approach 1: MediaPipe Pose & Self-Classifiers ||Minji Kim

In this work, we treat the problem of pose classification mainly by using the MediaPipe, where we employ landmarks functions to extract the x and y coordinates of all 33 key points in a body. The information related to key points is shown in (Figure 1). There are five different lists based on our targeted poses, and each pose list has x and y coordinates of the specific poses only.

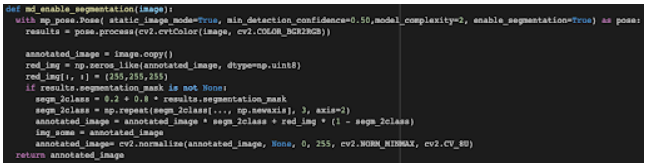
The five different lists have 66 inner lists, each of which represents both the x and y coordinates of key points. From the coordinate lists, the angles of a few selected key points are measured with a numpy.arctan2() function. Instead of calculating the angles of all points, we chose the necessary angles to be used for the next step: left and right shoulders, elbows, wrists, hips, knees, and ankles.

The selected angles are collected and put into a dictionary format with 12 keys in total. Because some poses, such as warrior2, require users to bend one knee while stretching the other, we decide to change left and right to less bent and more bent knee angles. A classifier in the method consists of multiple nested if statements where each nested if statement has a range of certain angles or coordinates. An image is used as an input, and the output label has the default value of "No Pose." If the image passes one of the nested statements, the label is changed into the following label. Based on our test dataset, the accuracies of downdog, tree, plank, warrior2, and goddess are 85.6%, 91.3%, 64.3%, 84.4%, and 80% respectively. The plank pose has much lower accuracy compared to other poses.

C.2. Approach 2: MediaPipe Pose & Classifiers ||Won Chung

As we noticed that we are directly applying the MediaPipe Pose into our model, we concluded that our model lacked a Machine Learning component necessary for developing an effective classification system, as we are only separating the angles. In order to implement more Machine Learning approaches, we've opted to train and evaluate our model utilizing existing classifiers. In order to properly train our model using the existing classifiers, we have determined that a significant amount of data cleaning methods are required.

First, to overcome the limitation of MediaPipe Pose detecting objects or humans in the background of images, which leads to the problem of achieving incorrect key points of the body, I've implemented a segmentation function to every image of the train dataset (Figure 2), where the function takes the original image (Figure 3) and blurs out the background of image (Figure 4). Then, to simplify our input procedure, I chose to develop a dictionary for both the train and test datasets. To successfully create a dictionary, I've created a function that takes images as an input, calculates the 12 body joint angles as in Approach 1 for each image, makes a list of the 12 body joint angles as a key to the dictionary, and assigns a label (poses: downdog, goddess, plank, tree, or warrior2) to the corresponding key. Due to the fact that dictionaries do not accept lists as keys, I transformed the list to a string in order to successfully assign joint angles as the key.



```
def md_enable_segmentation(image):
  with mp_pose.Pose( static_image_mode=True, min_detection_confidence=0.50,model_complexity=2, enable_segmentation=True) as pose:
    results = pose.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    annotated_image = image.copy()
    red_img = np.zeros_like(annotated_image, dtype=np.uint8)
    red_img[:, :] = (255,255,255)
    if results.segmentation_mask is not None:
      segm_2class = 0.2 + 0.8 * results.segmentation_mask
      segm_2class = np.repeat(segm_2class[..., np.newaxis], 3, axis=2)
      annotated_image = annotated_image * segm_2class + red_img * (1 - segm_2class)
      img_soma = annotated_image
      annotated_image= cv2.normalize(annotated_image, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
  return annotated_image
```

Figure 2. Enabling Segmentation and cleaning image

Figure 3.

Figure 4.

```
def making_dictionary:
    Using MediaPipe Pose:
    For every image in image files:
        Get joint angles
        Make list of joint angles
        Append new dictionary element with list
         as key and label as value
```

For our dataset, we have 1071 images for training and 470 images for testing, each with a distinguished pose. Us-

ing our dataset to create both training and test dictionaries, I've constructed the variables x train, y train, x test, and y test for classification reasons. The x train variable represented the keys of the train dictionary, which was a list of all the training images' joint angles. Consequently, the y train represented the labels corresponding to the x train, which was the matching pose. The x test variable represented the test dictionary's keys, whereas the y test variable represented the associated labels. As the keys were kept as strings, I had to use a for loop to convert the key back to a list and the elements of the list to float.

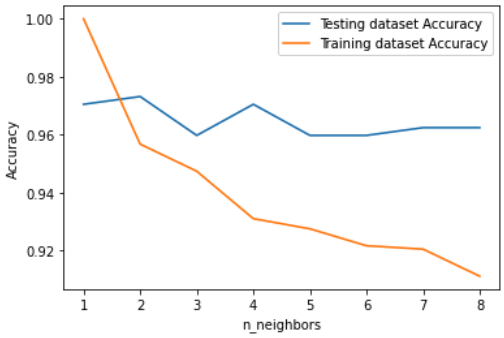


Figure 5.

For the evaluation purpose, I've opted to utilize the Logistic Regression Classifier as a baseline because it is the fastest training and predicting classifier and is suitable for the baseline. Then, I chose to test MLP, K-nearest neighbors classifier with K = 2 and 4. K was chosen to be 2 or 4 for K-nearest neighbors classifiers because these two values produced the highest testing accuracies among all K values between 1 and 8. (Figure 5). Below are the results:

Classifiers	**Confusion Matrix**	**Accuracy**
Logistic Regression	Figure 6	0.91
MLP	Figure 7	0.98
KNN (k=2)	Figure 8	0.97
KNN (k=4)	Figure 9	0.97

After obtaining the accuracies of all classifiers, there was a significant drawback to this method. Due to the lack of a dataset for incorrect postures, the model could only evaluate and categorize among the five distinct poses because it was not trained with the dataset for incorrect poses. Since there was no incorrect classification of pose, this may have contributed to the extremely high accuracy of the classifiers. Therefore, with this approach, a feasible improvement would be to collect people executing incorrect yoga poses and retrain the model with six poses, including the original five yoga poses and the incorrect pose. The model would then be better suited for actual applications, albeit with less accuracy.

Figure 6.

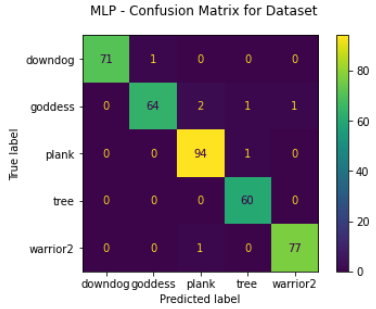


Figure 7.

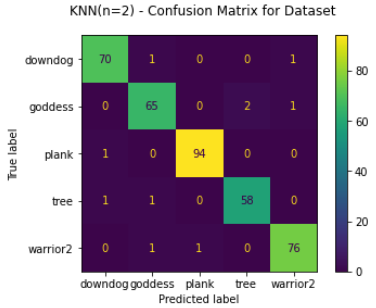


Figure 8.

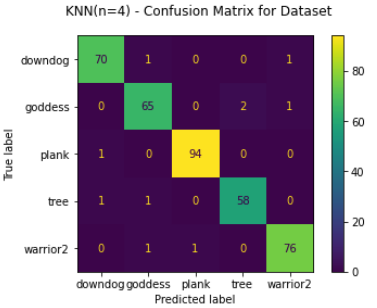


Figure 9.

C.3. Approach 3: Conv2D using Tensorflow ||Seung Hoon Jung

The problem with our initial model was that it lacked a machine learning algorithm that could manually train and test datasets as we input them. First, I tried to implement the Mediapipe Pose function so that we can have a trained model with custom datasets.

```
class PoseClassifier:

    def __init__(self, mode = False, upperBody = False, smooth=True, detectionCon = 0.5, trackCon = 0.5):

        self.mode = mode
        self.upBody = upBody
        self.smooth = smooth
        self.detectionCon = detectionCon
        self.trackCon = trackCon
```

Figure 10.

By using a custom PoseClassifier, I was initially attempting to develop a model that can detect a specific range of angles of the body for each yoga pose and have an error range of maybe plus or minus 5. The problem with this was that the dataset was not clean and needed to be consistent throughout so that the angles that I wanted corresponded with a particular part of the body.

I chose to develop a model that can take a live webcam video input and have it calculate the angles of joints while the person performs the yoga pose. If the angles fit inside the range of errors, then the model should be able to detect the pose. I used the following pseudocode:

```
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
```

However, this did not work with Google Colab and had to find another way of approaching the problem. There was also another problem of not having the key points aligned with the body exactly. This issue was fixed by changing the aspect ratio of the image and so that the key points would be aligned exactly where the joints were, hence the following code:

```
# Reshape image
    img = frame.copy()
    img = resized_img(dimension of (192, 192))
    input_image = tf.cast(img, dtype=tf.float32)
```

After repeating problems, I decided to implement a Keras model that can train the images from our dataset and train the dataset to obtain a raw accuracy and loss to see how accurate the predictions are for this model. The Keras Conv2D model takes in the number of filters the convolutional layers will learn from. It also takes in the kernel size to decide the dimensions of the 2D window.

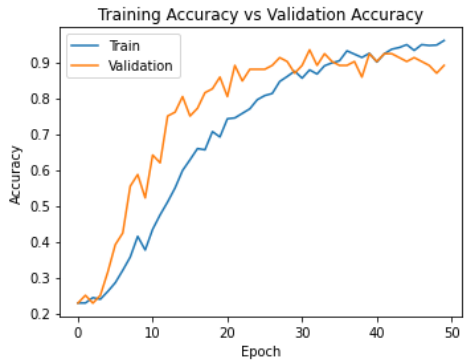


Figure 11.

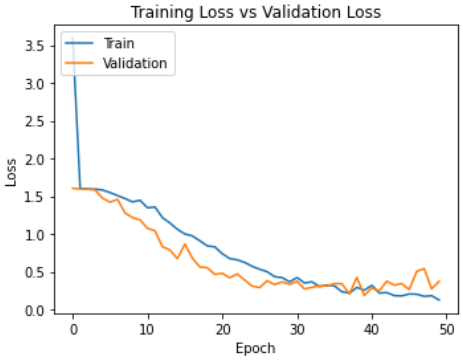


Figure 12.

```
train_loss, train_acc = model.evaluate(train_generator)
test_loss, test_acc   = model.evaluate(validation_generator)
print("final train accuracy = {:.2f} , validation accuracy = {:.2f}".format(train_acc*100, test_acc*100))
```

```
55/55 [==============================] - 100s 2s/step - loss: 0.0457 - accuracy: 0.9873
3/3 [==============================] - 11s 3s/step - loss: 0.3723 - accuracy: 0.8913
final train accuracy = 98.73 , validation accuracy = 89.13
```

Figure 13.

I loaded my dataset through colab and mounted it through my google drive. I categorized the data into train and test images and then generated tensor image data with normalized data with the 'flow_from_directory' in Keras. This categorizes the data into the following classes, which are my five yoga poses and counts the image count. I evaluated the model by calling fit(), so that the model trains over a specific amount of epochs, which in my case was 50. I wanted the model to train for an adequate amount so that the predictions came out more accurately. I return the loss values and metric values during training. (Figure 11, 12)

The next step was to evaluate the model on the test data and calculate the test loss and test accuracy. The accuracy of the training set came out to be 98 percent, which was a good average. Overall my method of predicting the accuracy of the model came out to be pretty accurate with an accuracy of 89.13 percent validation accuracy. (Figure 13)

D. Conclusion

Our pose detection model and classification was concise and limited as to the resources we had. We did not have a large dataset or set of records for us to evaluate our model completely in every circumstance, however, our implementations of the model produced a good output as well. Some future incorporations of this model could be to develop an automatic personal trainer that can tell people how they are doing during workouts and also a workout pose counter could develop even more just like the related works described above about the pushup counters. As pose trackers and keypoint connections and detecting gets more accurate, there would be even more developing applications going forward.

E. Contributions

- **Github Link: https://github.com/Davewchung/CVProject_CorrectPoseDetection**
  - **Won Chung**: MediaPipe Pose (Input vector dictionary) + Classifiers
  - **Seung Hoon Jung**: Keras Training Model + Data Collection
  - **Minji Kim**: MediaPipe Pose + Self Classifier
- **Won Chung**
  - Method and Experiment for Approach 2
  - Introduction
- **Seung Hoon Jung**
  - Method and Experiment for Approach 3
  - Conclusion
- **Minji Kim**
  - Method and Experiment for Approach 1
  - Related Works

References

[1] Chronic disease fact sheet: Physical inactivity, Sep 2022. 1

[2] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition - researchgate. 1

[3] Google. Google/mediapipe: Cross-platform, customizable ml solutions for live and streaming media. 1

[4] Aryan Vij. Push-ups with python! (mediapipe + open, Feb 2022. 1