International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 05 Issue: 06 | June - 2021 ISSN: 2582-3930

HUMAN POSE DETECTION AND ESTIMATION

Dr. ML Sharma C[1] | Mr. Vinay Kumar Saini[2] |Jai Raj Singh[3]

Department of IT Engineering ,Maharaja Agrasen Institute of Technology,New Delhi

India

Abstract

As you might guess, pose estimation is a pretty complex issue: humans come in different shapes and sizes; have many joints to track and are often around other people and/or objects, leading to visual occlusion. We want our machine learning models to be able to understand and smartly infer data about all these different bodies.

Pose Estimation is one of the more elegant applications of neural networks and is startlingly accurate and sometimes, seems like something right out of science fiction.

1.Introduction

Pose estimation refers to computer vision techniques that detect human figures in images and video, so that one could determine, for example, where someone's elbow's position shows up in an image. To be clear, this technology is not recognizing who is in an image there is no personal identifiable information associated to pose detection. The algorithm is simply estimating where key body joints are. Pose estimation has many uses, from interactive installations that react to the body to augmented reality, animation, fitness uses, and more. We hope the accessibility of this model inspires more developers and makers to experiment and apply pose detection to their own unique projects. While many alternate pose detection systems have been open-sourced, all require specialized hardware and/or cameras, as well as quite a bit of system setup.

Typically, working with pose data means either having access to special hardware or having experience with C++/Python computer vision libraries.

As you might guess, pose estimation is a pretty complex issue: humans come in different shapes and sizes; have many joints to track (and many different ways those joints can articulate in space); and are often around other people and/or objects, leading to visual occlusion. Some people use assistive devices like wheelchairs or crutches, which may block the camera’s view of their bodies; others might not have certain limbs; and still others may have very different proportions. We want our machine learning models to be able to understand and smartly infer data about all these different bodies.

In the past, technologists have approached the problem of pose estimation using special cameras and sensors (like stereoscopic imagery, mocap suits, and infrared cameras) as well as computer vision techniques that can

extract pose estimation from 2d images (like OpenPose). These solutions, while effective, tend to require either expensive and not widely distributed technology, and/or familiarity with computer vision libraries and C++ or Python. This makes it harder for the average developer to quickly get started with playful pose experiments.

Pose Estimation is one of the more elegant applications of neural networks and is startlingly accurate and sometimes, seems like something right out of science fiction.For Instance, check out Google's Move Mirror, an in-browser application that estimates the user's pose in real time and then displays a movie still with the actor holding the same pose.

2.Related work

Pose Estimation is a general problem in Computer Vision where we detect the position and orientation of an object. This usually means detecting keypoint locations that describe the object.

For example, in the problem of face pose estimation (a.k.a facial landmark detection), we detect landmarks on a human face. We have written extensively on the topic. Please see our articles on (**Facial Landmark Detection using OpenCV** and **Facial Landmark Detection using Dlib**)

A related problem is **Head Pose Estimation** where we use the facial landmarks to obtain the 3D orientation of a human head with respect to the camera.

In this article, we will focus on human pose estimation, where it is required to detect and localize the major parts/joints of the body (e.g. shoulders, ankle, knee, wrist etc.).

Remember the scene where Tony stark wears the Iron Man suit using gestures?

If such a suit is ever built, it would require human pose estimation!

Approach

2.1 Keypoint Detection Datasets

Until recently, there was little progress in pose estimation because of the lack of high-quality datasets. Such is the enthusiasm in AI these days that people believe every problem is just a good dataset away from being demolished. Some challenging datasets have been released in the last few years which have made it easier for researchers to attack the problem with all their intellectual might.

Some of the datasets are :

1. **COCO Keypoints challenge**
2. **MPII Human Pose Dataset**
3. **VGG Pose Dataset**

If we missed an important dataset, please mention in the comments and we will be happy to include in this list!

3.Multi-Person Pose Estimation model

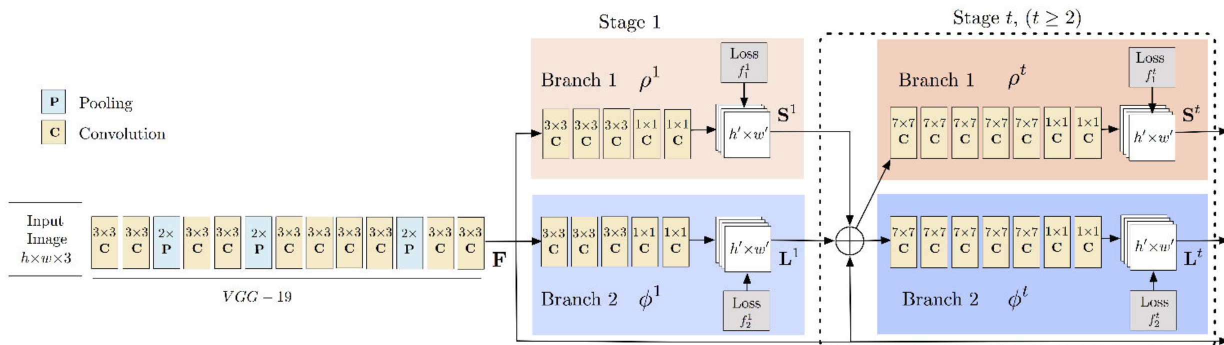
The model used in this tutorial is based on a paper titled Multi-Person Pose Estimation by the Perceptual Computing Lab at Carnegie Mellon University. The authors of the paper train a very deep Neural Networks for this task. Let's briefly go over the architecture before we explain how to use the pre-trained model.

3.1. Architecture Overview

The model takes as input a color image of size w × h and produces, as output, the 2D locations of keypoints for each person in the image. The detection takes place in three stages :

1. **Stage 0**: The first 10 layers of the VGGNet are used to create feature maps for the input image.

2. **Stage 1**: A 2-branch multi-stage CNN is used where the first branch predicts a set of 2D confidence maps (S) of body part locations (e.g. elbow, knee etc.). Given below are confidence maps and Affinity maps for the keypoint – Left Sh oulder.



3.2 Pre-trained models for Human Pose Estimation

The authors of the paper have shared two models – one is trained on the Multi-Person Dataset (MPII) and the other is trained on the COCO dataset. The COCO model produces 18 points, while the MPII model outputs 15 points. The outputs plotted on a person is shown in the image below.

COCO KeyPoints

MPII KeyPoints

COCO Output Format Nose – 0, Neck – 1, Right Shoulder – 2, Right Elbow – 3, Right Wrist – 4, Left Shoulder – 5, Left Elbow – 6, Left Wrist – 7, Right Hip – 8, Right Knee – 9, Right Ankle – 10, Left Hip – 11, Left Knee – 12, LAnkle – 13, Right Eye – 14, Left Eye – 15, Right Ear – 16, Left Ear – 17, Background – 18 **MPII Output Format** Head – 0, Neck – 1, Right Shoulder – 2, Right Elbow – 3, Right Wrist – 4, Left Shoulder – 5, Left Elbow – 6, Left Wrist – 7, Right Hip – 8, Right Knee – 9, Right Ankle – 10, Left Hip – 11, Left Knee – 12, Left Ankle – 13, Chest – 14, Background – 15

4. Code for Human Pose Estimation in OpenCV

In this section, we will see how to load the trained models in OpenCV and check the outputs. We will discuss code for only single person pose estimation to keep things simple. As we saw in the previous section that the output consists of confidence maps and affinity maps. These outputs can be used to find the pose for every person in a frame if multiple people are present. We will cover the multiple-person case in a future post.

First, download the code and model files from below. There are separate files for Image and Video inputs. Please go through the README file if you encounter any difficulty in running the code.

4.1. Step 1 : Download Model Weights

Use the getModels.sh file provided with the code to download all the model weights to the respective folders. Note that the configuration proto files are already present in the folders.

From the command line, execute the following from the downloaded folder.

1 sudo chmod a+x getModels.sh

./getModels.sh

Step 2: Read Image and Prepare Input to the Network

2 The input frame that we read using OpenCV should be converted to a input blob (like Caffe) so that it can be fed to the network. This is done using the blobFromImage function which converts the image from OpenCV format to Caffe blob format. The parameters are to be provided in the blobFromImage function. First we normalize the pixel values to be in (0,1). Then we specify the dimensions of the image. Next, the Mean value to be subtracted, which is (0,0,0). There is no need to swap the R and B channels since both OpenCV and Caffe use BGR format.

4. Step 3: Make Predictions and Parse Keypoints

Once the image is passed to the model, the predictions can be made using a single line of code.

The **forward** method for the DNN class in OpenCV makes a forward pass through the network which is just another way of saying it is making a prediction.

RESULTS

-Importing Images :

```
In [28]: import cv2
         import time
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
         from PIL import Image
         import mpi as mpi
         from numpy import set_printoptions
         from sklearn.preprocessing import Normalizer
         import math

In [72]: image1 = 'test_images/test3.jpeg'
         image2 = 'user_images/user_img3.jpeg'

In [73]: from IPython import display
         display.Image(image1)

Out[73]:
```

```
In [74]:
         display.Image(image2)
```

-Adding Common Frames to the Images :

```
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)
npoints=15
POSE_PAIRS = [[0,1], [1,2], [2,3], [3,4], [1,5], [5,6], [6,7], [1,14], [14,8], [8,9], [9,10], [14,11], [11,12], [12,13]]
```

```
In [76]: # Read image
frame = cv2.imread(image1)

# Specify the input image dimensions
inWidth = 368
inHeight = 368

# Prepare the frame to be fed to the network
inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight), (0, 0, 0), swapRB=False, crop=False)

# Set the prepared object as the input blob of the network
net.setInput(inpBlob)
```

```
In [77]: out = net.forward()
```

```
In [78]: H = out.shape[2]
W = out.shape[3]
# Empty list to store the detected keypoints
points = []
for i in range(len(points)):
    # confidence map of corresponding body's part.
    probMap = output[0, i, :, :]

    # Find global maxima of the probMap.
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    # Scale the point to fit on the original image
    x = (frameWidth * point[0]) / W
    y = (frameHeight * point[1]) / H

    if prob>threshold and amp and gt :
        cv2.circle(frame, (int(x), int(y)), 15, (0, 255, 255), thickness=-1, lineType=cv.FILLED)
        cv2.putText(frame, "{}".format(i), (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1.4, (0, 0, 255), 3, lineType=cv2.LINE_AA)

        points.append((int(x), int(y)))
    else :
        points.append(None)

# cv2.imshow("Output-Keypoints",frame)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

- Normalizing the coordinates :

```
In [79]: list1=[]
list2=[]
for pair in POSE_PAIRS:
    partA = pair[0]

    partB= pair[1]
    list1.append(partA)
    list2.append(partB)
image1_points = mpi.findpairs(list1,list2,image1)
```

```
In [80]: image1_points
Out[80]: [[140.5614316813639, 68.2741739997029],
 [243.64320651511682, 62.32699725453415],
 [134.77171355623972, 69.44659001647958],
 [299.65245928234737, 88.07592740349948],
 [456.527359375388, 85.27347841420544],
 [473.7775754256571, 85.46580665459612],
 [453.6014568709755, 96.37036180989084],
 [341.6734507510895, 99.10652020931963],
 [347.74631931310876, 105.2114726066579],
 [485.8024517405241, 144.05076175770665],
 [482.6812708368918, 135.5801676742465],
 [495.7678894408099, 313.08906307147606],
 [478.84381723584687, 219.2741736992457],
 [310.61995645183913, 270.14993880336874],
 [445.2137492368313, 256.64538146840016]]
```

```
In [81]: Data_normalizer = Normalizer(norm='l2').fit(image1_points)
Data_normalized_1 = Data_normalizer.transform(image1_points)
```

```
In [82]: print(Data_normalized_1)
[[0.89950444 0.43691162]
 [0.96880302 0.247832  ]
 [0.88892424 0.45805426]
 [0.95941502 0.28199791]
 [0.98299887 0.18361163]
 [0.98411595 0.1775269 ]
 [0.9781676  0.2078176 ]
 [0.96041318 0.27857947]
 [0.95715128 0.28958839]
 [0.95873934 0.28428661]
 [0.96274131 0.27042406]
 [0.84551008 0.53395947]
 [0.90920567 0.41634728]
```

-Finding out the cosine similarity:

```
In [88]: image1_pointsA=[]
         for i in Data_normalized_1:
             #image1_pointsA.append(i[0])
             image1_pointsA.append(i[1])

         image2_pointsB=[]
         for i in Data_normalized_2:
             #image2_pointsB.append(i[0])
             image2_pointsB.append(i[1])

         sum_points=0
         for i in range(len(image2_pointsB)):
             sum_points=sum_points+(image1_pointsA[i]*image2_pointsB[i])
         #print(sum_points)

         sum_squaresA=0
         sum_squaresB=0
         for i in range(len(image1_pointsA)):
             sum_squaresA=sum_squaresA+(image1_pointsA[i]**2)
             sum_squaresB=sum_squaresB+(image2_pointsB[i]**2)
         #print((sum_squaresA**(1/2))*(sum_squaresA**(1/2)))
         print("Cosine Similarity :",sum_points/((sum_squaresA**(1/2))*(sum_squaresA**(1/2)))-0.03)

         Cosine Similarity : 0.9397947111440924

In [89]: def slope(list1):
             sample=[]
             for i in range(len(list1)):
                 for j in range(i+1,len(list1)):
                     if(i==j):
                         sample.append(0)
                     else:
                         z=(list1[i][1]-list1[j][1])/(list1[i][0]-list1[j][0])
                         sample.append(z)
             return sample

In [90]: slope_image1=slope(Data_normalized_1)
         print(slope_image1)

         [-2.728477359905188, -1.9983205243646227, -2.5857488068501078, -3.033735334902267, -3.0655962098507077, -2.91234209305802, -2.5
```

CONCLUSION

During the current times of covid when people are bound to stay at home, and health is the major concern it becomes more important to stay fit and active. With the increasing trend of exercising at home, it becomes important to ensure that the poses we are doing during our workout are perfect as wrong poses may have an adverse impact on the body . Here comes our model to solve this problem of incorrect postures, the model once trained can be used as a personal trainer to check the exercise poses.