Report:
Mesh Recovery for 3d Human Pose Estimation Model

CS256 Group C

Abstract

For our project we are designing and training an application that involves the computer vision of pose estimation method. Pose Estimation is when computer vision detects the human's position and orientation of an object. The simple approach is when a person's detector comes first, this follows by estimating the parts and then calculating the pose for each person. The deep learning-based 3D human pose and mesh estimation are based on human pose and shape parameters of their models such as SMLP and MANO from an input image. 3D human pose estimation finds the three parameters (X, Y, Z coordinates) of an object from the 2D object pose estimation. After looking through the State of the Art (SOTA) approach, we chose the Posenet and Meshnet model and decided how we can use it for the project. Pose2mesh uses a graph convolutional system to recover 3D human pose and mesh from the 2D human pose. We show that our pose2mesh performs better than the previous 3D pose estimation methods in different datasets.

Introduction

3D human pose estimation is ever advancing - using deep learning computer vision, it can predict human body spatial positioning from a two dimensional image. This ability is helpful in progressing fields like media/gaming - CGI special effects - and for human applications - like detailed body precision feedback.

Prominent 3D pose estimation methods face significant issues with domain(controlled data vs in-the-wild data) and rotation variances. 3D pose estimation finds the X, Y, and Z coordinates of the object from an image, but with mesh modeling it can take a step further to create a 3D structure of the object, seemingly similar to dense pose estimation but focusing on detection vs the accuracy of constructing the object shapes. This 3D structure that we get from the mesh modeling helps alleviate the domain issue. Mesh modeling takes all of the faces, edges, and vertices of an object shape to compile into the 3D structure of the object. Our project goal is to improve upon current state-of-the-art 3D human pose estimation methods that use mesh modeling to both further understand and progress detection and estimation capabilities.

The paper that inspired our project provided a top implementation of a bi-network method to gain highly accurate 3D human pose results and provided Pytorch code implementation - and our goal was to both delve deeper in understanding the paper's implementation choices while also testing differences to determine if we might be able to improve it. We also test

the Pose2Mesh's robustness by creating our own data set from our web crawler to test the accuracy and check for overfitting.

Literature Survey

Pose2Mesh is a very recent state-of-the-art 2D pose-based approach, model-free mesh method for 3D human pose estimation. It simultaneously detects where the mesh vertices and human joint points are from a 2D pose by implementing PoseNet and then MeshNet together with a 4to get highly accurate detection results.

In "Pose2Mesh: Graph Convolutional Network for 3D Human Pose and Mesh Recovery from a 2D Human Pose", we learned about how the graphCNN based Pose2Mesh architecture worked. Using a graphCNN, which like it sounds is a CNN that takes a graph as input, allows us to effectively take in the 2D pose estimation as the input for the Pose2Mesh's first 'stage', the PoseNet.

In a slightly older state-of-the-art paper, "Convolutional Mesh Regression for Single-Image Human Shape Reconstruction", they also used a graphCNN, model-free, approach and outlined the issues with the then popular optimization-based shape recovery used for 3D pose estimation. Optimization-based approaches could achieve great accuracy, but were highly sensitive to the initialization and local minimas. The graphCMR used in this second paper was able to assuage those issues - like most learning-based approaches have been able to do in their own ways - by pipelining an image-based CNN, with an initial encoder, to a graph CNN to get the whole 3D geometry of the human pose while removing the need for a parametric space that is pre-specified.

Pose2Mesh similarly pipelines the PoseNet and MeshNet to get a 3D human body pose from the original by using a graphCNN in the MeshNet. However, Pose2Mesh results are better defined, cleaner, 3D outputs compared to the graph CMR even though Pose2Mesh uses the 2D pose as input versus the original image itself like in other learning-based approaches. This saves time and greatly reduces the domain and rotational issues that approaches like graphCMR had still faced.

To achieve similar and mostly better results than the previously leading 3D estimation mesh approaches, pose2mesh also normalized the 2D input pose since the 3D pose is independent of the location and scale of the input, and synthesized errors in the 2D input poses to better represent real world input (errors that would happen with occlusion, for example).

Methodology

The current 3D pose estimation methods are categorized into two different approaches: an image-based and a 2D pose-based approach. The

image-based approach takes an RGB image as an input for 3D pose estimation and 2D pose-based approach takes the 2D human into the 3D space. Our approach follows the 2D pose approach to make Pose2Mesh more robust as the difference between the training set and the testing set.

PoseNet is used to synthesize errors on the 2D pose. The estimated 2D input pose often contains errors, especially the challenging poses. We train the PoseNet by minimizing the loss distance function between the predicted 3D pose and the groundtruth.

MeshNet concatenates both 2D and 3D pose into graph convolution on pose. There are different ways to represent 3D data such as volumetric grid, multi-views, and mesh data. MeshNet uses four loss functions: Vertex, joint, surface normal, and surface edge loss.

Datasets:

Human3.6 is a 3D body pose based indoors which consists of 3.6M video frames.

Coco is an in-the-wild dataset with different 2D annotations like detection and human joints.

These datasets will be discussed in the results section.

Implementation details

As part of this project, the pose2mesh network was used to achieve the human pose estimation model. The pose2mesh consists of two parts. The first is the posenet model which takes in the input in MS COCO format. A 2D pose from a photo is converted to a 3D pose in the posenet. The implementation provides various parameters for the user to adjust based on the requirement and system configuration. To get familiarized with the project implementation, we ran a posenet model on the Human3.6 dataset. The first challenge for this run was the disk size for the amazon ec2 instance. The original implementation stores every epoch information on the disk. This was the root cause of high memory usage. One of the tasks for the project was to understand the reason for storing every epoch of information on the disk. To train the pose2mesh model, the original implementation only makes use of a single epoch weight from posenet training. To achieve the best weight and in turn best pose2mesh performance, the implementation compares results from each posenet epoch. This way of comparing and storing epoch data probably consumed most of the disk space. For the last milestone, we have implemented a deletion part of the posenet training. This design will ensure that while posenet training is ongoing, only two epoch data is stored at the maximum. At the end of the training, only the best result epoch from the training will be present in the experiment directory.

The second challenge for the posenet was the resumption of training from the last run epoch. The original implementation did not support the resumption of the posenet training by using the stored data from previously

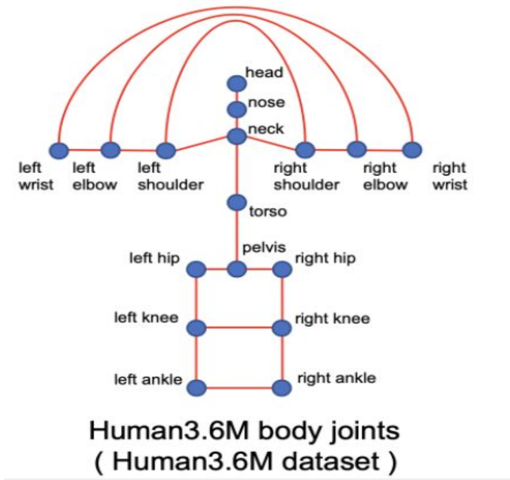
run epochs. This restriction caused a significant delay in our milestones as each training would take 2-3 days to complete and if something goes wrong in between, the time spent till that failure would be a waste. The original implementation uses a yaml configuration file approach to feed the model hyperparameters. These yaml files are used by posenet as well as pose2mesh model. For supporting the resumption of posenet training, we added a new field in the yaml files to load the directory where the previous run result is stored. We also made the code change to support the use of a pre-trained dataset for posenet training. It is important to point out that the original implementation already had the support to use pre trained posenet results to train the pose2mesh model.

For comparing the results and to understand the choices made by the original implementation, we decided to use different optimizer functions. The RMSPROP was used for the original implementation. The optimizer functions are used by both posenet and pose2mesh models. We replaced RMSPROP with Adam optimizer for both models. The comparison of the results will be covered further in the next sections. Lastly, we had to adjust the model hyperparameters for pose2mesh as the CUDA memory shortage was hit during the training. The original implementation had set batch size as 64 which was replaced by 32 in our implementation. A significant downside for this change was the exponential increase in the runtime of each epoch. An epoch for pose2mesh takes about 30-40 hours.

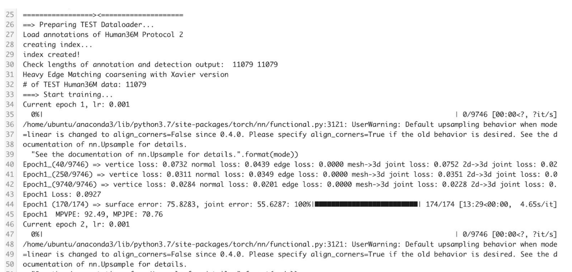
Lastly, through this implementation, we have achieved the posenet and pose2mesh training with the Human3.6 dataset. Pre-trained posenet model data with different datasets were used to train the pose2mesh model. Pose2mesh was trained with a combination of images from Human3.6, COCO, and MuCO datasets. The validation is done with a PW3D dataset.

Results and Discussion

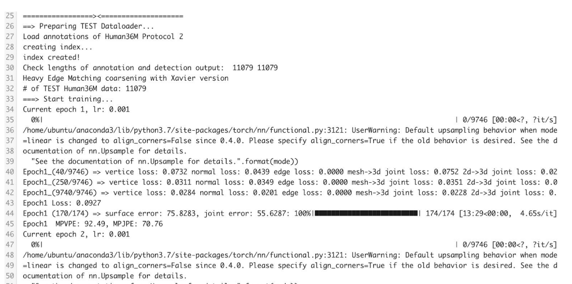
Posenet and pose2mesh model is trained using Human3.6M dataset. It represents 2d human pose containing joints as shown in image. Dataset is represented in COCO format. It contains annotations like object detection, keypoint detection, stuff segmentation,panotic segmentation, densepose and image captioning as in JSON format. Posenet identifies keypoints with certain confidence score between 0 and 1. It lifts 2d pose to 3d pose. That is used as an input to pose2mesh to generate mesh.



Here is snippet of human Pose2Mesh model Having Adam optimizer for 15 epochs.



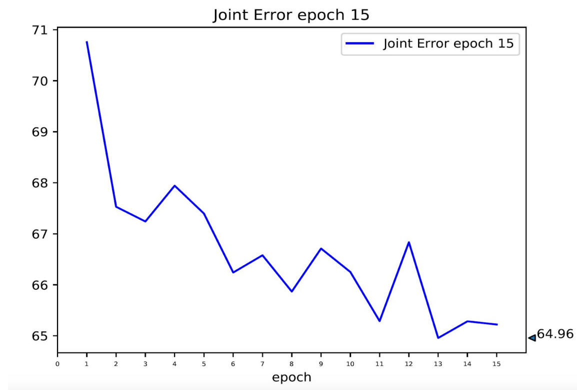
```
=================><====================
==> Preparing TEST Dataloader...
Load annotations of Human36M Protocol 2
creating index...
index created!
Check lengths of annotation and detection output:  11079 11079
Heavy Edge Matching coarsening with Xavier version
# of TEST Human36M data: 11079
===> Start training...
Current epoch 1, lr: 0.001
  0%|                                                          | 0/9746 [00:00<?, ?it/s]
/home/ubuntu/anaconda3/lib/python3.7/site-packages/torch/nn/functional.py:3121: UserWarning: Default upsampling behavior when mode
=linear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the d
ocumentation of nn.Upsample for details.
  "See the documentation of nn.Upsample for details.".format(mode))
Epoch1_(40/9746) => vertice loss: 0.0732 normal loss: 0.0439 edge loss: 0.0000 mesh->3d joint loss: 0.0752 2d->3d joint loss: 0.02
Epoch1_(250/9746) => vertice loss: 0.0311 normal loss: 0.0349 edge loss: 0.0000 mesh->3d joint loss: 0.0351 2d->3d joint loss: 0.0
Epoch1_(9740/9746) => vertice loss: 0.0284 normal loss: 0.0201 edge loss: 0.0000 mesh->3d joint loss: 0.0228 2d->3d joint loss: 0.
Epoch1 Loss: 0.0927
Epoch1 (170/174) => surface error: 75.8283, joint error: 55.6287: 100%|████████████████████| 174/174 [13:29<00:00,  4.65s/it]
Epoch1  MPVPE: 92.49, MPJPE: 70.76
Current epoch 2, lr: 0.001
  0%|                                                          | 0/9746 [00:00<?, ?it/s]
/home/ubuntu/anaconda3/lib/python3.7/site-packages/torch/nn/functional.py:3121: UserWarning: Default upsampling behavior when mode
=linear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the d
ocumentation of nn.Upsample for details.
```

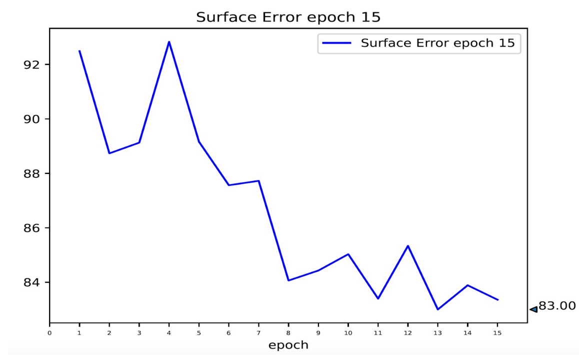


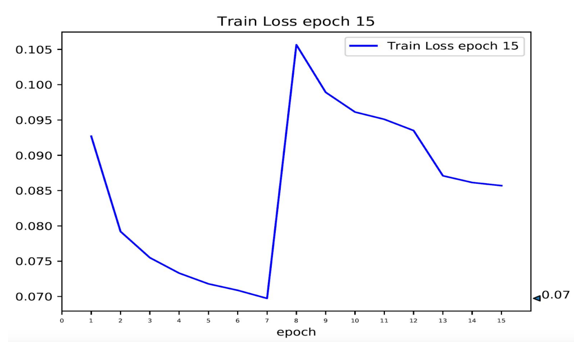

```
=================><====================
==> Preparing TEST Dataloader...
Load annotations of Human36M Protocol 2
creating index...
index created!
Check lengths of annotation and detection output:  11079 11079
Heavy Edge Matching coarsening with Xavier version
# of TEST Human36M data: 11079
===> Start training...
Current epoch 1, lr: 0.001
  0%|                                                          | 0/9746 [00:00<?, ?it/s]
/home/ubuntu/anaconda3/lib/python3.7/site-packages/torch/nn/functional.py:3121: UserWarning: Default upsampling behavior when mode
=linear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the d
ocumentation of nn.Upsample for details.
  "See the documentation of nn.Upsample for details.".format(mode))
Epoch1_(40/9746) => vertice loss: 0.0732 normal loss: 0.0439 edge loss: 0.0000 mesh->3d joint loss: 0.0752 2d->3d joint loss: 0.02
Epoch1_(250/9746) => vertice loss: 0.0311 normal loss: 0.0349 edge loss: 0.0000 mesh->3d joint loss: 0.0351 2d->3d joint loss: 0.0
Epoch1_(9740/9746) => vertice loss: 0.0284 normal loss: 0.0201 edge loss: 0.0000 mesh->3d joint loss: 0.0228 2d->3d joint loss: 0.
Epoch1 Loss: 0.0927
Epoch1 (170/174) => surface error: 75.8283, joint error: 55.6287: 100%|████████████████████| 174/174 [13:29<00:00,  4.65s/it]
Epoch1  MPVPE: 92.49, MPJPE: 70.76
Current epoch 2, lr: 0.001
  0%|                                                          | 0/9746 [00:00<?, ?it/s]
/home/ubuntu/anaconda3/lib/python3.7/site-packages/torch/nn/functional.py:3121: UserWarning: Default upsampling behavior when mode
=linear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the d
ocumentation of nn.Upsample for details.
```


Each snippet consists of vertical loss,Normal Loss, Edge loss, 3d joint loss, 2d joint loss, Surface error and joint eror.

Following are graphs generated while training Pose2mesh for joint error, surface error and train loss for each epoch.







Below is the sample output file generated at the end of Pose2Mesh training. One file contains 3d coordinates and other file contains corresponding mesh.

```
v -0.19456962 -0.35008702 -0.5979098
v -0.1968442 -0.33334473 -0.5915373
v -0.18307397 -0.3362985 -0.5886799
v -0.17809543 -0.35123062 -0.5909624
v -0.18572228 -0.3250443 -0.58404475
v -0.19733286 -0.3219664 -0.58333015
v -0.17100495 -0.33759472 -0.58177376
v -0.16450924 -0.3499318 -0.58098495
v -0.1527749 -0.3165316 -0.5535629
v -0.14424244 -0.31895024 -0.5457325
v -0.14768189 -0.3282131 -0.55302
v -0.1556512 -0.3241616 -0.5587798
v -0.1450668 -0.29818094 -0.53370976
v -0.15191534 -0.28494635 -0.53285825
v -0.14307277 -0.28430188 -0.52200717
v -0.13734263 -0.29750848 -0.5221618
v -0.19072676 -0.28314993 -0.5398991
v -0.1828436 -0.27812102 -0.538854
v -0.18156803 -0.2838995 -0.5442222
v -0.18880035 -0.28852963 -0.54504585
v -0.20426452 -0.28634936 -0.54682016
v -0.20286027 -0.29262328 -0.5488477
v -0.2091614 -0.2919748 -0.55255073
v -0.20948192 -0.28491366 -0.550973
v -0.19300126 -0.27799973 -0.53478074
v -0.18550044 -0.27375096 -0.53199095
v -0.19564733 -0.28572085 -0.54213285
v -0.19786896 -0.2809859 -0.5404102
v -0.20351994 -0.27001616 -0.5427848
v -0.20276676 -0.27410594 -0.5454284
```

Method	MPJPE	PA-MPJPE
Lassner et al.		93.9
HMR	88.0	56.8
NBF		59.9
Pavlakos et al.		75.9
Kanazawa et. al.		56.9
GraphCMR		50.1
Arnab et al.	77.8	54.3
SPIN		41.1
Pose2Mesh(RMSPROP)	64.9	46.3
Pose2Mesh(Adam)	60.2	47.2

Here various state of arts of 3d pose estimation through MPJPE and PAMPJPE error matrix. MPJPE is mean per joint position error. PAMPJPE is procrustes analysis mean per joint position error. Here is the formula how MPJPE is calculated.

$$MRPE = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{R}^{(i)} - \mathbf{R}^{(i)*}||_2,$$

Here we notice that Pose2Mesh with adam optimizer has low MPJPE value than Pose2Mesh with RMSPROP. Infact it has the lowest MPJPE value compared to other state of arts. But it

has higher PA-MPJPE value than Pose2Mesh with RMSPROP.

References

[1] https://arxiv.org/pdf/2008.09047.pdf

Pose2Mesh: Graph Convolutional Network for 3D Human Pose and Mesh Recovery from a 2D Human Pose, August 2020

[2] https://arxiv.org/pdf/1905.03244.pdf

Convolutional Mesh Regression for Single-Image Human Shape Reconstruction, May 2019