3D Human Pose Estimation with Mesh Recovery

Group C
CS256 Milestone 4

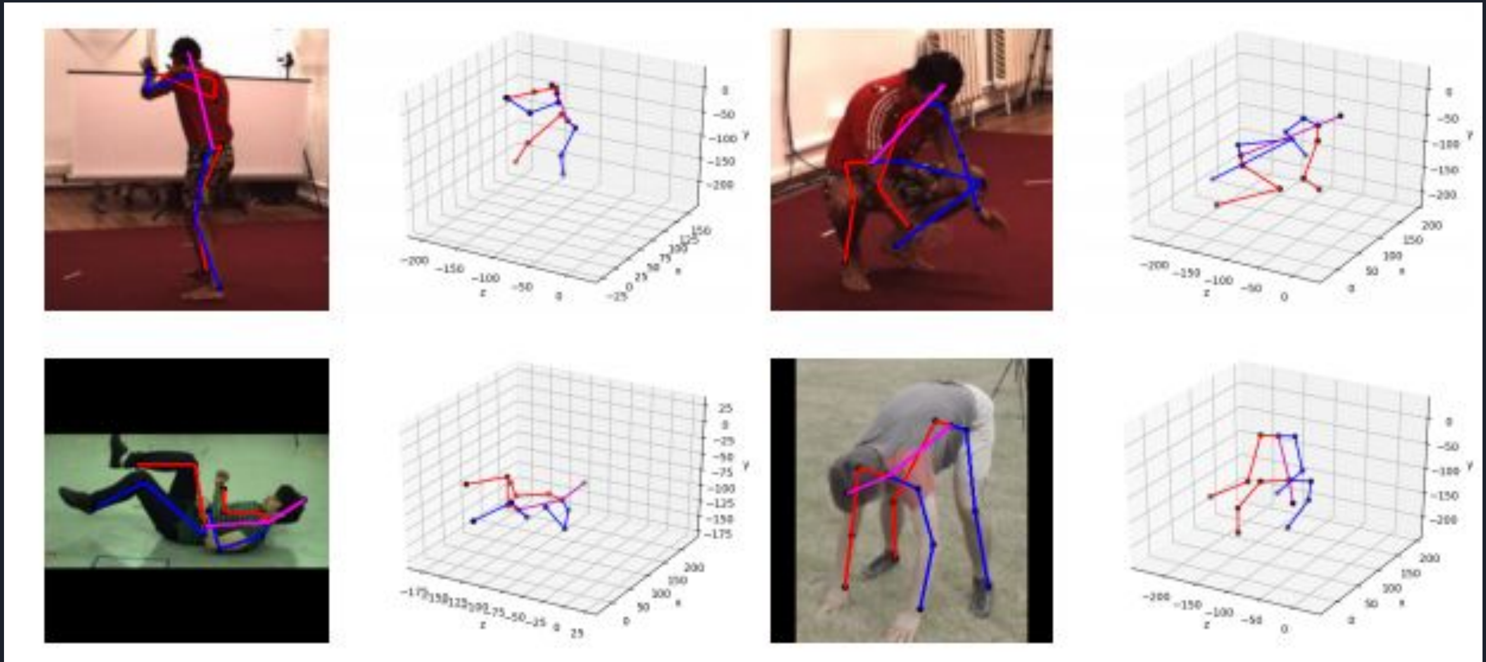
Project Idea: 3D human pose estimation

- 3D pose estimation is the task of producing a 3D pose that matches the spatial person.
- 3D can predict human body position from 2D.
- Problem statement: 3D pose estimation finds the X, Y, Z coordinates of the object from the image and predicts the identical pose.

- Solution: improve state-of-the-art 3d pose estimation methods that uses mesh modeling to further understand, detect, and estimate capabilities.

3D Human Pose estimation

Image classification



Pose2Mesh

- State of art
- 2-d coordinates of joints to Mesh
- Contains 2 parts
  a. Posenet
     - Takes 2-d input of joints .
     - Lifts 2-d image to 3-d image
  b. Meshnet
     - Genererates mesh as output.
- Graph CNN
- RMSPROP optimizer vs ADAM optimizer

Literature Survey

“Pose2Mesh: Graph Convolutional Network for 3D Human Pose and Mesh Recovery from a 2D Human Pose”

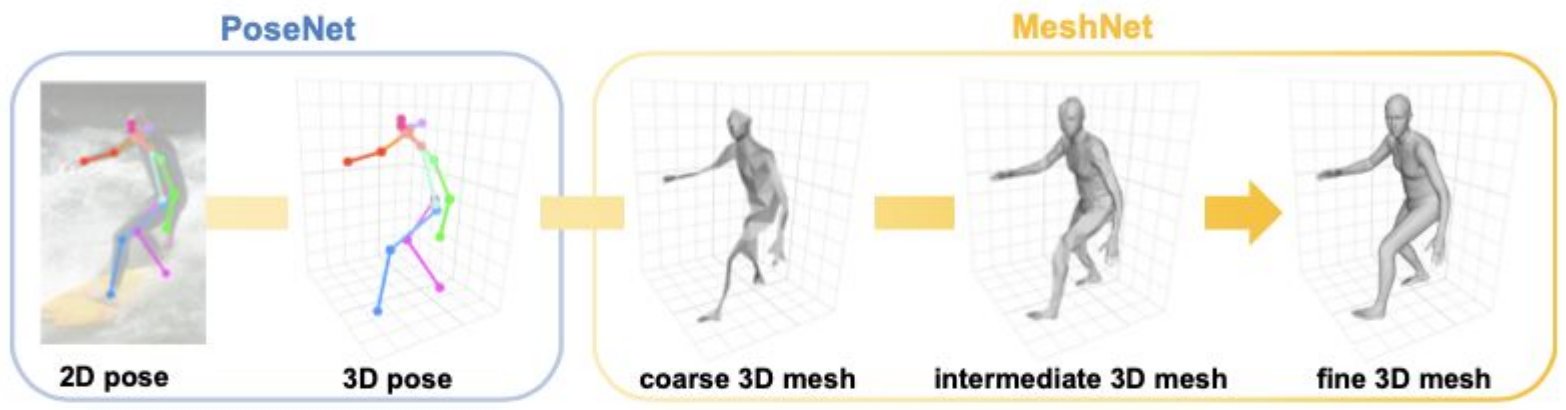


Fig. 1: The overall pipeline of Pose2Mesh.

Literature Survey Cont. & SOTA Progress

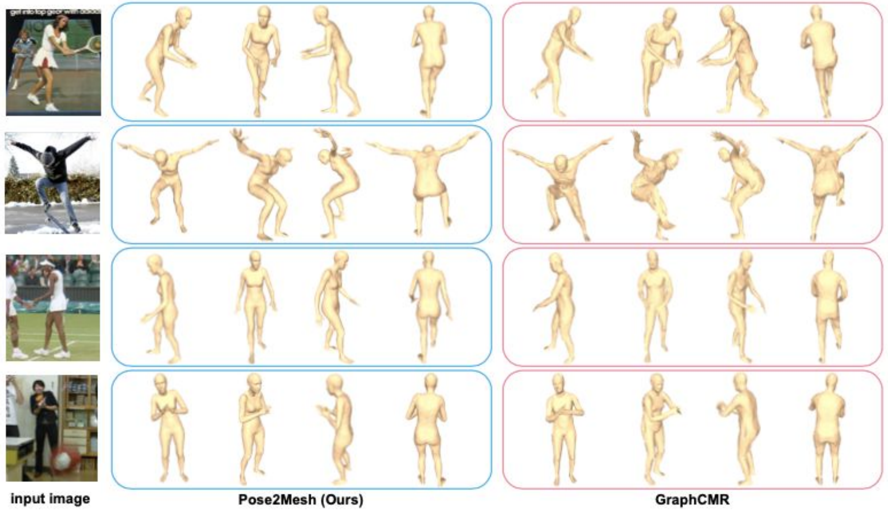


Fig. 7: The mesh quality comparison between our Pose2Mesh and GraphCMR [31].

Pose2Mesh is leading in mesh quality compared to past approaches.

Convolutional Mesh Regression for Single-Image Human Shape Reconstruction, 2019 (GraphCMR)

Data Set - web crawler

- Data we need here is human images
- Huge datasets are readily available in the market
  - HUMAN 3.6M, COCO SMPL etc
- Data follows MS COCO format - JSON Annotation format

```
Human36M_subject*_data.json
|-- 'images': [
                {'id': image id,
                'file_name': image file name,
                'width': image width,
                'height': image height,
                'subject': subject id,
                'action_name': action name,
                'action_idx': action id,
                'subaction_idx': subaction id,
                'cam_idx': camera id,
                'frame_idx': frame id
                },
                 ...,
                {same dict}
                ]
|-- 'annotations': [
                {'id': annotation id,
                'image_id': id of image where this annotations belongs to,
                'bbox': bounding box (xmin, ymin, width, height)
                },
                …,
                {same dict}
                ]

Human36M_subject*_camera.json
|-- camera id: {
                'R': 3x3 rotation matrix (extrinsic parameter),
                't': 3-dimensional translation vector in milimeter (extrinsic parameter),
                'f': 2-dimensional focal length (x- and y-axis) in pixel (intrinsic parameter),
                'c': 2-dimensional principal point coordinates (x- and y-axis) in pixel (intrinsic
        parameter)
                }

Human36M_subject*_joint_3d.json
|-- subject id
   |-- action id
      |-- subaction id
         |-- frame id: 17x3 joint coordinates in world coordinate system (not camera-centered
coordinate system, you need to multiply camera extrinsic matrix to transform it to
```

Web Crawler

- BeautifulSoup - Python library for parsing HTML and XML docs
- Demo
- Annotations

Implementation Details

- Planned activities and implementations:
  - Achieve higher accuracy
  - Run on Google Colab
  - Use of different loss and activation functions
  - Employ our own dataset

- Achieved milestones:
  - Run on AWS EC2
  - Use of different optimization function
  - Use of pre-trained models
  - Improve original implementation for efficient memory usage.
  - Support resumption of training with previously generated models.

AWS EC2

- To setup EC2 instance, we had to run requirement.sh as well as manual installation of python libraries.
- To begin with, the instance had around 150GB of hard disk which quickly ran out as every epoch data from the training was stored.
- We increased the hard disk size twice to finally have 350GB. Most of the space was consumed by the datasets.
    - A single dataset can take up to 150GB.
    - In the end, we ended up having three datasets on the instance
- CUDA device memory issue was hit for pose2mesh training.
    - A limit was hit for the max memory usage on CUDA device.
    - Tackled the issue by decreasing the batch size.

Optimization function

- We planned to use different optimization functions than RMSPROP which was used in the original implementation.
- Adam optimizer is widely used across industry as a standard optimization function.
  - Most of the hyperparameters for the model are provided by the YAML file.
  - Code to handle different optimization functions was added as part of this implementation.
- Comparison for the results from RMSPROP vs. Adam will be explored further in the result section.

Pre-trained models

- As discussed previously, there are two models in this design.
    - Posenet
    - Meshnet - Pose2Mesh
- The program supports the use of pre-trained posenet models to train the pose2mesh model.
- Due to CUDA device memory restriction, batch size was reduced which in turn increased the run time for each epoch.

```
Epoch1 Loss: 0.1047
Epoch1 (500/555) => surface error: 99.7129, joint error: 75.6595: 100%|██████████████████████████| 555/555 [42:48<00:00,  4.63s/it$
Epoch1  MPVPE: 116.71, MPJPE: 97.53
Current epoch 2, lr: 0.001
  0%|                                                                                                | 0/31186 [00:00<?, ?it/s$
/home/ubuntu/anaconda3/lib/python3.7/site-packages/torch/nn/functional.py:3121: UserWarning: Default upsampling behavior when mod$
=linear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the $
ocumentation of nn.Upsample for details.
  "See the documentation of nn.Upsample for details.".format(mode))
Epoch2_(31100/31186) => vertice loss: 0.0238 normal loss: 0.0258 edge loss: 0.0000 mesh->3d joint loss: 0.0215 2d->3d joint loss:
Epoch2 Loss: 0.0975
Epoch2 (500/555) => surface error: 101.1725, joint error: 76.8685: 100%|█████████████████████████| 555/555 [42:14<00:00,  4.57s/it$
Epoch2  MPVPE: 112.63, MPJPE: 92.57
Current epoch 3, lr: 0.001
```

Efficient memory usage

- The original implementation stored data/weights from each epoch.
  - There is a concept of “best” epoch result.
  - The results are compared from each epoch to select the best epoch data.
- In deep learning, previous weights are used to learn and improve current epoch performance.
- The hard disk memory limitation forced us to implement efficient memory usage from the model training runs.
- Our implementation compares the result from previous epoch to achieve the best data.
  - Only store if the result is improved from previous epoch.
  - Freed up almost 60GB memory.

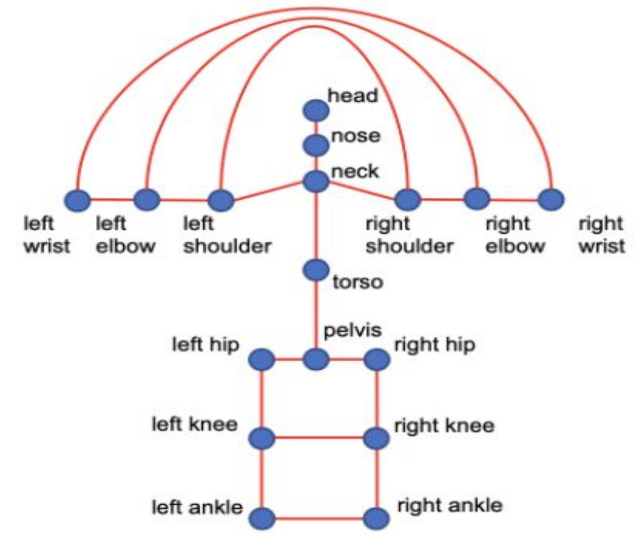
Resumption of training

- Original implementation was making use of pre-trained posenet model to train the pose2mesh model.
    - The same technique can be extended for continuation of the posenet training.
- As described previously, we faced a lot of issues at the start of the project.
- After each failure, we would start the training from scratch which cost us a great amount of time.
    - Each training run takes time in range of days.
- With this change, we are now able to resume posenet and pose2mesh training from previously stored result.

Demo

Result

- Human3.6M dataset for posenet and pose2mesh training.
- COCO format. Annotation for
    - Object detection
    - Keypoint detection
    - Stuff segmentation
    - Panoptic segmentation
    - Densepose
    - Image captioning
- JSON format



Human3.6M body joints
(Human3.6M dataset)

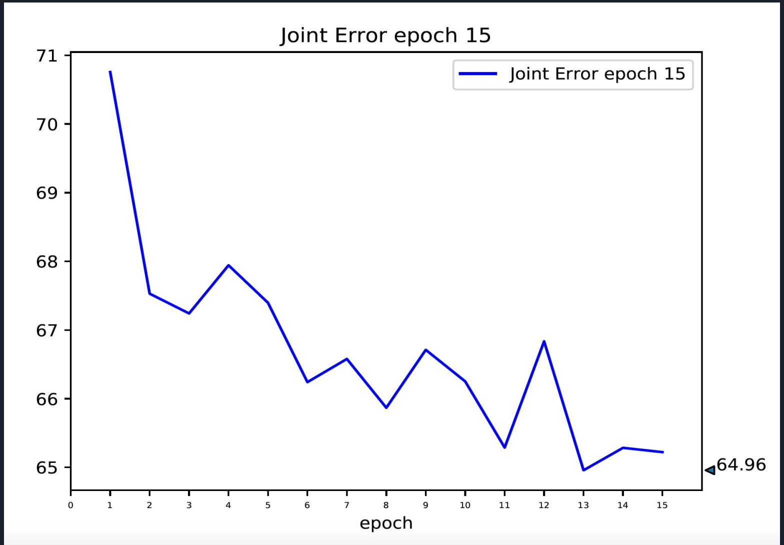
Results Contd..

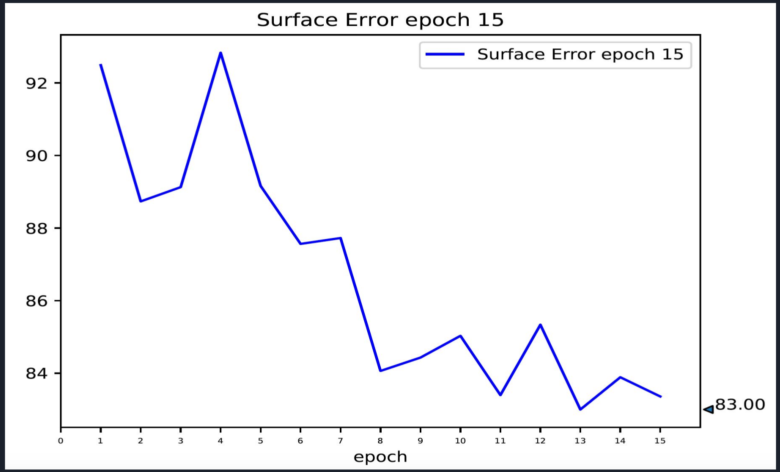
- Snipet of 15 epochs of Pose2Mesh model on Human3.6 dataset using Adam optimizer.
- Following things are calculated
    - Vertice loss
    - Normal loss
    - Edge loss
    - 3d joint loss
    - 2d joint loss
    - Surface error
    - Joint error

```
=================><===================
==> 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.
```

Result Contd..





Result Contd..

```
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
```

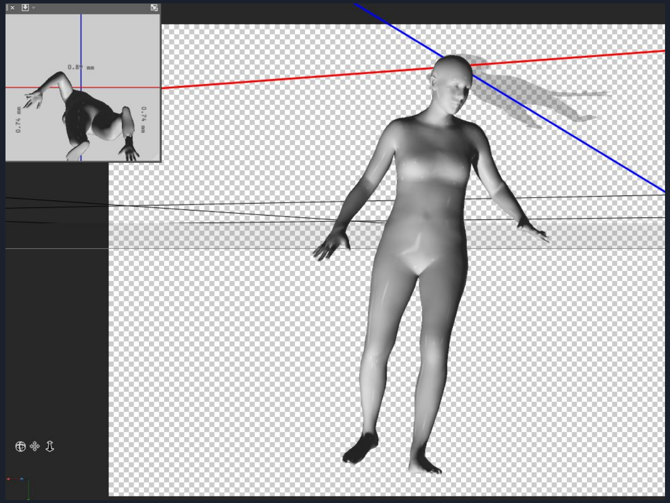
Method	MPJPE	PA-MPJPE
Lassner et al.		93.9
HMR	88.0	56.8
NBF		59.9
Pavlakos et al.		75.9
Kanazawa et. al.		56.9
GraphCMR		50.1
Arnab et al.	77.8	54.3
SPIN		41.1
Pose2Mesh(RMSPROP)	64.9	46.3
Pose2Mesh(Adam)	60.2	47.2

MPJPE -Mean Per Joint Position Error

PAMPJPE – Procrustes Analysis Mean Per Joint Position Error

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

Conclusion: CG Implementation



A Computer Graphics Approach to tweak/reevaluate Pose2Mesh Outputs

1. Supporting Rigid Skin Geometry
   a. a correct placement of vertices/triangle faces on its fingers and toes

Future Work

1. Pose Estimation Translation to Joint Chains
   a. PoseNet's 3d pose intact with final outputs
   b. Bounding Boxes for which vertice belong to which bone
   c. Open OpenFrameworks/Maya Scripts => run Smooth Skinning Algorithm on c file => AutoRiggingModel
2. Texture Mapping
   a. Account for the image's categorical traits to render a basic texture for that body
   b. Displacement/Bump Mapping for skin pores and better muscle tone generation