CIS 581: Computer Vision & Computational Photography Fall 2021

3D Human Pose Estimation From RGB Images

Members: Jasleen Kaur Dhanoa, Manasa Sathyan, Ruchi Gupte, Sharon Richu Shaji, and Swati Gupta

1 Objective

This project aims to generate visually intuitive 3 Dimensional Pose estimations for humans from 2D RGB images. Most 3D pose estimation solutions require the use of 3D datasets, and a method that can convert 2D images to 3D key joint positions using only a 2D dataset requires additional algorithms and methods to accomplish the task. We achieve this by breaking down our project into three intermediate steps, namely (i) 2D key-point generation, (ii) conversion of 2D to 3D poses using Generative Adversarial Networks (GANs), and (iii) rendering the generated 3D poses for visually informative results.

2 Overview and Related Work

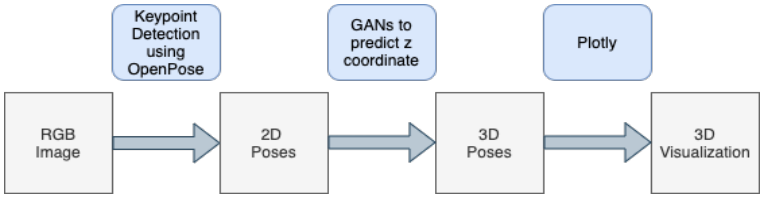


Figure 1: Project Pipeline

In order to handle the task of obtaining a 3D pose from a single 2D image, we plan to divide the given task into modules as shown in the pipeline (Fig 1).

Module 1: Key-point Generation
First, we detect the joint positions (key-points) for full-body human poses and generate the 2D pose. We explored Deep Learning algorithms such as DensePose [1] and OpenPose [2] for this task. Pose Detection by both models can be seen in Fig (2)
After experimentation with both pose estimation models we decided to go ahead with OpenPose. The outputs received from this model better suited our pipeline and hence gave better overall results. OpenPose was also much less computationally expensive than DensePose.

Module 2: 2D to 3D Pose Conversion Here, we convert the 2D pose estimation we get from the previous step into 3D poses. We have used a Generative Adversarial Network (GANs) based approach [3] to generate the z-coordinate estimate for each joint (effectively getting the 3D Pose). The driving idea for the GANs model is that a correctly estimated 3D pose should not become discontinuous even when rotated in 3D space, so its projection back in 2D plane should be valid 2D pose too. Thus the discriminator for the GANs model aims to discriminate between the "real" and the generated "fake" 2D poses. Additionally, by making the model account for unlikely joint directions and limb constraints using a heuristic, we are able to generate 3D joint positions which present naturally and realistically for humans in 3D space.

Module 3: Rendering and Visualisation Next, we render these generated 3D joint positions to 3D space and visualise the limb and body joint positions as 3D stick figures. This step allows us to obtain visually informative and intuitive results, and enables us to sanity check the proposed pipeline on the images on the test set. This allows us to fine tune parameters and redefine certain constraints based on the visuals, giving us an accurate and refined 3D pose estimation model.
Paper [3] served as our primary inspiration for this project. [4], [5], [9] helped inform us of our approach for the project. We referenced [1], [2] while deciding on our keypoint detection method. [3], [10] were referenced while formulating and coding up our GAN model. Finally, we surveyed [6], [7], [8] to choose the best rendering/ visualization tool for our data.

3 Dataset

The 2D image dataset used to train and test the model for our project was the MPII dataset [4], which is a state-of-the-art benchmark for evaluation of articulated human pose estimation. It consists of 25K images and contains over 40K humans performing over 400 different activities. This gives us a variety of images to train on, with one or multiple people in the frame, along with poses where joint positions overlap or are presented differently, allowing us to test the robustness of the GANs model.
Open Source Code/Dataset(s) Used: *2D to 3D using GANs*, *OpenPose*, *Plotly*, *MPII*

4 Methodology

Key-point Generation using OpenPose: Using OpenPose, a CNN based open-source 2D Pose Estimation system, we are able to extract full-body human pose key-points from single RGB Images. This model was used to convert all the 2D images from the **MPII Human Pose Dataset** to 2D joint locations. We feed the trained OpenPose model single-person and multi-person RGB images from the MPII Dataset and save two different forms of output - (i) RGB Images with visual skeleton overlaid Fig.(2), Fig.(3), and (ii) JSON files containing key-points i.e, (x,y) coordinates for each of the 25 joints in the human body for each human detected. Open pose is able to identify and classify the various people in each frame as separate entities and each json file contains a nested dictionary with people and their respective joint positions. We parse these .json files and extract 2D (x,y) coordinates which are fed as input into the next section of our pipeline.

(a) Input RGB Image (b) Output RGB Image with Skeletons Overlaid

Figure 2: Single-person Pose Estimation

Pre-processing OpenPose Results: Before the data can be used as input for training, we need to normalize the joint positions and remove joints that do not benefit the model.

(a) Input RGB Image (b) Output RGB Image with Skeletons Overlaid

Figure 3: Multi-person Pose Estimation

First, while the OpenPose model provided 25 joint positions in its 2D pose estimation, joints such as ears and eyes did not contribute to the pose of the human being. So, we decided not to incorporate these joints while training. This assumption allowed for a more efficient and accurate model and gave significantly better results for pose estimations. Hence, the final dataset used as the input for the GAN model consists of only the first 15 joints.
Second, the 2D joint positions for each input pose had to be normalized. This is a crucial step for 3D pose estimation as it reduces the losses and inconsistencies that could be generated due to differences in coordinates of the people in the images and the scale. The paper [3] details a method for normalisation that works well for the human pose estimation problem. To implement this, we first select a joint to be the center point for normalisation, in our case: the central hip joint. For each pose we first subtract the position of each joint by the central hip joint, and then divide this difference by Euclidean mean of all the differences. This gives a normalised output centered around the central hip joint.
Finally we split this dataset into training and test datasets, and train our GANs model on the training dataset.

Framework of GAN Architecture

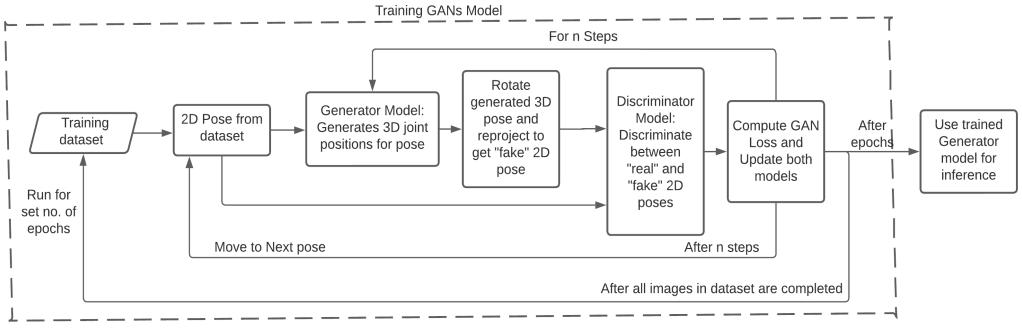


Figure 4: GAN Pipeline

Figure 5 shows the detailed framework used to train the GANs model. We coded the GAN architecture for the 2D to 3D pose estimation problem in PyTorch from scratch based on the approach shared in the original paper. Given the pre-processed input of the 2D joint key-points (i.e the (x,y) values for each of the 15 joints in the human body), we implemented Multi-Layer Perceptron based networks for both the Generator and the Discriminator and use the standard GAN loss function to back-propagate and gradient descent.

The Generator first predicts fake z-coordinates for a given 2D pose (pairs of x-y coordinates for each joint). These z-coordinates are combined with their respective 2D joint positions to represent the "predicted" 3D joint positions for the pose, i.e the x-y-z coordinates of the joints positions. This predicted 3D pose is then rotated with respect to the y axis by a randomly sampled θ angle, and is then projected back to the 2D (x-y) plane to obtain a new 2D pose. This "fake" 2D pose is then fed to the discriminator along with the real 2D pose to predict fake/real labels. The goal of the discriminator is to accurately distinguish real vs fake (i.e minimize the binary cross entropy loss) while the goal of the generator is to fool the discriminator (i.e. maximize the discriminator's loss on fake 2D poses). Thus, at the end of various training loops and epochs, we get a trained model in which the generator is able to convincingly convert 2D poses to 3D.
The Generator model uses 3 hidden linear neural network layers and a LeakyReLU activation function. The output layer size of the Generator is the same as the number of joints i.e. 15. Similarly, the Discriminator model has 3 hidden linear neural network layers, uses a LeakyReLU activation function along with a sigmoid function. The output layer size in this model is set to 1. To get convergence and a well trained generator model, we allowed the generator to take multiple learning steps for each discriminator step.

Loss Function Used:
The main loss function used for the generator and discriminator is the Binary Cross Entropy Loss. Since the output of the discriminator is binary, predicting whether the human pose given is Real or Fake, the binary cross entropy loss alone is used. On the other hand, for the generator, we have implemented an additional heuristic loss function. This was due to the problem of inverted 3D poses that were obtained from the generator for certain test cases. The discriminator is unable to differentiate between the wrong 3D poses, whose z-component is inverted, from the correct 3D poses since the projection of correct 3D pose after rotating by θ is the same as the projection of the wrong 3D pose after rotating by $-\theta$.
The heuristic loss function is calculated as follows: We consider the face orientation vector as $f = [f_x, f_y, f_z] = v_{nose} - v_{neck}$ where v_{nose} and v_{neck} are the 3D joint locations of the nose and neck respectively. We also consider the shoulder orientation vector as $s = [s_x, s_y, s_z] = v_{ls} - v_{rs}$ where v_{ls} and v_{rs} are the 3D joint locations of the left shoulder and right shoulder respectively.
Now, we calculate $\sin\beta = \frac{f_z s_x - f_x s_z}{||f||||w||}$ where, β is the angle between f and s on the zx-plane and must satisfy the condition $\sin\beta >= 0$. We introduce the following additional loss function:

$$L_{angle} = max(0, -\sin\beta)$$

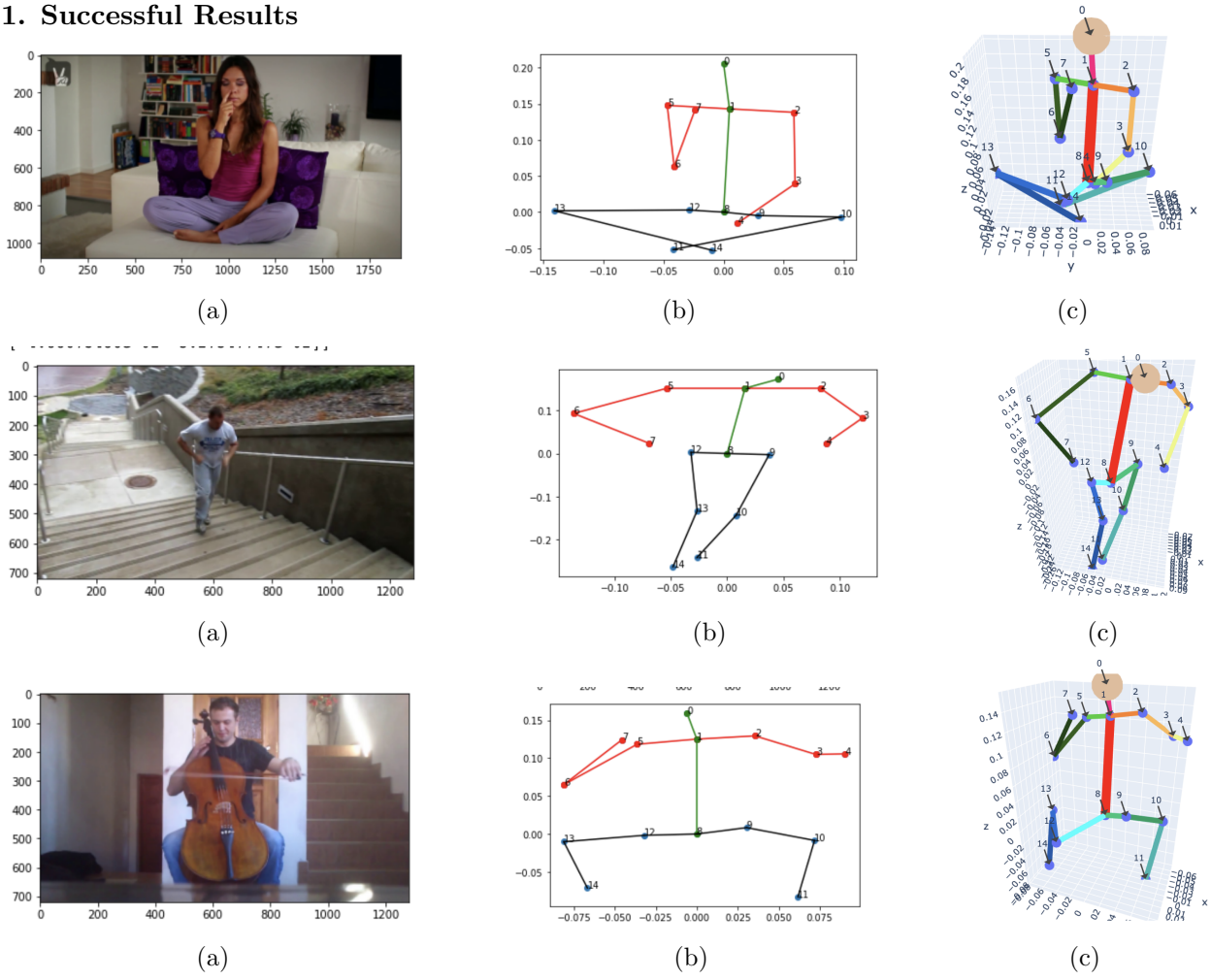
The final combined loss function for the generator is as follows:

$$Loss(G, D) = BCELoss + L_{angle}$$
$$Loss(G, D) = \mathbb{E}\, p[logD(p)] + \mathbb{E}\, p, \theta[log(1 - D(f(p, G(p); \theta)))] + L_{angle}$$

3D Visualization using Plotly
After obtaining the 3D Pose containing the (x,y,z) coordinate estimates of the detected keypoints, we wanted to visualise it. Initially we tried doing 3D visualisation using Matplotlib, however the generated plot did not have interactive control and it was difficult to visually inspect if the joint positions and poses were correct. Therefore, we decided to go ahead with Plotly for 3D Visualisation. It allows interactive control of the plot and the ability to view the same pose from different angles which is very useful while determining if the given pose is correct. We wanted to render 3D volumetric models of the Poses, however the approaches such as DenseRaC[6] and Pixel2mesh[7] directly use images to generate a 3D mesh, optimise it and generate 3D volumetric renderings. Since, they don't incorporate 3D keypoints (generated by our GAN model) to inform the rendering it would be an interesting future work, but was out of scope for this project.

5 Results and Discussion

1. Successful Results



(a) (b) (c)

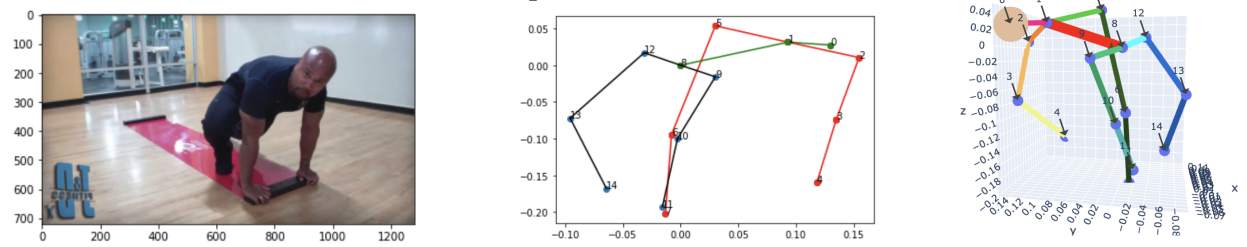
(a) (b) (c)

(a) (b) (c)

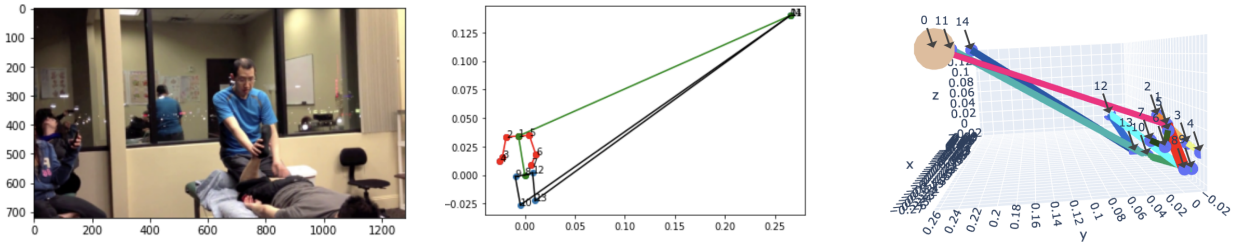
Figure 7: Qualitative results on the MPII Dataset. Column (a) denoted the original image, column (b) denotes the 2D pose estimation using OpenPose, and column (c) denotes the 3D pose estimated using the proposed GANs model.

The results above show that the GANs architecture built learns the data well and outputs close to accurate 3D pose estimations. We have added some more examples of successful results in Appendix 7.1

2. Failed Results due to incorrect OpenPose estimates



(a) (b) (c)

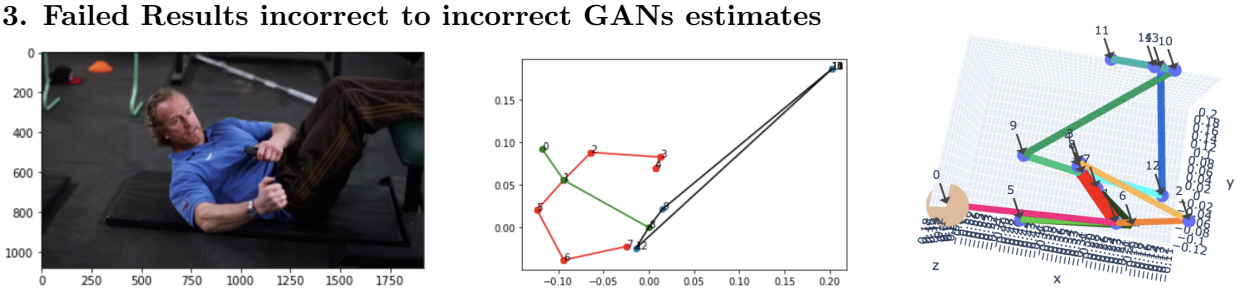


(a) (b) (c)

Figure 9: Column (a) denoted the original image, column (b) denotes the 2D pose estimation using OpenPose, and column (c) denotes the 3D pose estimated using the proposed GANs model.

From the results above, we can clearly see that the 2D pose estimation by OpenPose did not give any meaningful results. Note that despite this, the GANs model was able to estimate a 3D pose very similar to what one would expect from the given 2D pose.

3. Failed Results incorrect to incorrect GANs estimates



(a) (b) (c)

Figure 10: Column (a) denoted the original image, column (b) denotes the 2D pose estimation using OpenPose, and column (c) denotes the 3D pose estimated using the proposed GANs model.

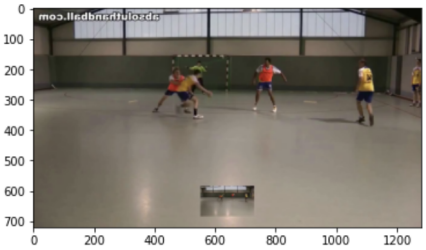
From the original image above, we can see that the part of the legs below the knee is cut out. This is correctly captured in the OpenPose result. However, in the 3D pose estimated by our GANs model, the part below the knee is also estimated even though it was not present in images (a) and (b), giving an incorrect output. This test case has not been tackled yet in our implementation and we hope to incorporate a solution to this soon.

6 Conclusion and Future Work

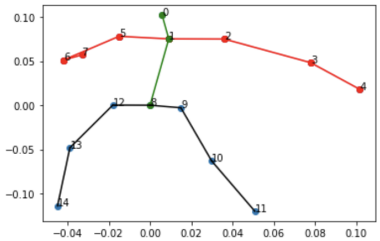
The GANs model for 3D pose estimation was able to give us fairly accurate and robust results verified by visualizing the generated 3D poses. By accounting for human pose constraints in the loss function, the model was able to avoid generating inverted hands and legs in the final pose estimate. While the GANs model worked effectively for most situations, there were still some errors in creating a proper estimation when one or more joints was not in the frame/ not detected by the OpenPose 2D estimator. One possible future modification would be to add a workaround for these cases to ensure that a few points don't affect the 3D model generation of the whole. The current Pose Detection represents and estimates it pose in terms of key-points and stick figures, but some 3D Pose detectors use Volumetric representations of the human pose, which has great use in robotics applications. That is why a possible future project would be to create a model that could represent and create 3D Volumetric posed from 2D RGB Images

7 Appendix

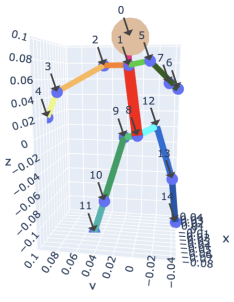
7.1 Results of GAN Model



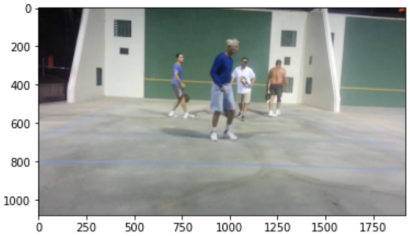
(a)



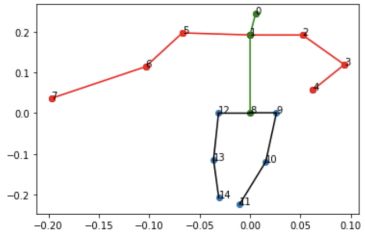
(b)



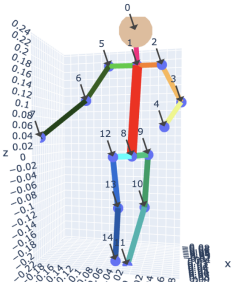
(c)



(a)

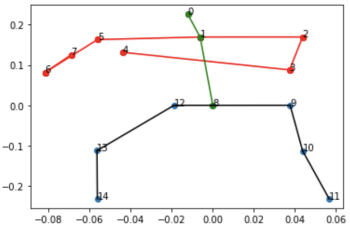


(b)

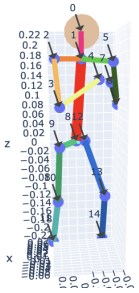


(c)

(a)



(b)



(c)

7.2 DensePose vs OpenPose

(a) DensePose Output

(b) OpenPose Output

Figure 14: Pose Estimation Models for Key point Generation

7.3 References

1. Güler, R.A., Neverova, N. and Kokkinos, I., 2018. Densepose: Dense human pose estimation in the wild. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7297-7306).

2. Cao, Z., Hidalgo, G., Simon, T., Wei, S.E. and Sheikh, Y., 2019. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. IEEE transactions on pattern analysis and machine intelligence, 43(1), pp.172-186.

3. Kudo, Y., Ogaki, K., Matsui, Y. and Odagiri, Y., 2018. Unsupervised adversarial learning of 3d human pose from 2d joint locations.

4. Mykhaylo Andriluka and Leonid Pishchulin and Peter Gehler and Schiele, Bernt, 2014. 2D Human Pose Estimation: New Benchmark and State of the Art Analysis, IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

5. Zheng, C., Wu, W., Yang, T., Zhu, S., Chen, C., Liu, R., Shen, J., Kehtarnavaz, N. and Shah, M., 2020. Deep learning-based human pose estimation: A survey. arXiv preprint arXiv:2012.13392.

6. Xu, Yuanlu, Song-Chun Zhu, and Tony Tung. "Denserac: Joint 3d pose and shape estimation by dense render-and-compare." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

7. Wang, Nanyang, et al. "Pixel2mesh: Generating 3d mesh models from single rgb images." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

8. Varol, Gul, et al. "Bodynet: Volumetric inference of 3d human body shapes." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

9. Tome, Denis, Chris Russell, and Lourdes Agapito. "Lifting from the deep: Convolutional 3d pose estimation from a single image." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

10. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).