Human Pose Estimation using Deep Learning

Ashish Ranjan Jha
EPFL
ashish.jha@epfl.ch

Abstract

This report summarizes our findings in human 3D pose estimation task from images using deep convolutional neural network. We train our deep network to regress 3D poses from 0.15 million different images of the Human36M dataset. The regressor network is pre-trained by training a body part detection network and then using the learned weights for initializing the pose regression network. We test our results on approximately 13,000 images. On testing, we achieve promising results. The results are further improved by using optimizations such as weight decay, momentum, dropout, and a non-linear activation at the last regression layer of the regression network.

1 Introduction

The human pose estimation problem is widely known in the field of computer vision, and various machine learning models applied to this problem provide promising results [4]. A step further, with the use of a larger dataset, this problem can be approached by more powerful models like deep convolutional neural networks. This report discusses the estimation of human 3D poses from 2D grayscale images in a discriminative approach, where the pose estimation task is viewed as a regression problem. At a high level, the aim is to extract features from the 2D images and then learn a mapping from the feature space to the 3D pose space. We use a deep convolutional neural network as our machine learning model.

For the deep learning framework to learn the patterns of human pose from image data, we require a large dataset for training. Human36M [9], the dataset that we have used, provides 3.6 milion video frames with labeled poses of several human subjects performing various tasks. This huge size of data makes it possible for deep networks to train effectively. In our present work, we restrict to using only 'Walking' action, which is one of the seventeen available tasks in the dataset, for both training and testing. The raw data from the dataset requires fair amount of pre-processing for both the images as well as the corresponding poses. The images are available as video frames, hence we need to perform operations like cropping, background subtraction, image compression and RGB to grayscale conversion before actually feeding them as inputs to the deep networks. Similarly, the 2D and 3D pose data is processed equivalently to match the modified images. Overall, we use approximately 0.15 million images as input for training and 0.01 million images for testing our results.

Convolutional neural networks [8] are widely used today for vision problems because of less number of parameters (as compared to a fully-connected deep neural network), which reduces overfitting and makes training easier along with the added advantage of having translation invariance.Inspired by [1], in our approach to train deep convolutional networks, we first pre-train the network using human-body parts detection task and then train the main pose regression network using the learned weights. The pre-training helps in better initialization of weights as compared to a random initialization. Our final network contains three convolutional and six fully connected layers, three each for detection and regression tasks. We wrote a CPU implementation in Theano [5], of the convolution task and all operations inherent in training convolutional neural network, which are available along with all the pre-processed 128 X 128 grayscale image as well as 2D and 3D pose data at the EPFL CV lab server.[1]. Theano codes are also avaliable at github. [2]

2 Related Work

The problem of human pose estimation has been approached by a wide variety of machine learning methods. [4] gives an overview on various pose estimation methods. In [6] a cascade neural network is used having 3 stages for estimating 2D human pose from RGB images. In each stage, the network architecture is similar to the network in [2], although it is applied to joint point prediction in 2D images. Networks in the later stages take higher resolution input windows around the previous predictions to refine the previous predictions.

In [7] random forests are used to estimate the body part labels from depth images and given the predictions of the part labels, mean shift is applied to obtain the part locations. Such works have demonstrated also that learning body part labels adds on to achieve better features for pose estimation. [3] trains a multi-task deep convolutional network for 2D human pose estimation, consisting of the pose regression and body part detection tasks. All tasks share same convolutional feature layers, and the regression network is shown to benefit from sharing features with the detection network. A similar approach is taken up in our work, but for 3D poses. Until now, deep convolutional networks have mostly been used for classification tasks [2]. Given sufficient data, these deep networks can learn decent features for regression tasks from even randomly initialized weights [1].

3 The Dataset

We use the Human36M dataset [9] for this project. Human36M is a dataset containing 3.6 million 3D human poses and corresponding images. The image data is generated by recording videos for seventeen different actions like walking, greeting, eating, etc. performed by eleven subjects, six male and five female, from 4 different camera angles with two sub-actions for each action. The pose data is recorded by a MoCap (motion capture) system. In the dataset, we have RGB images, which we convert to grayscale images. For the human body part detection network, we use the 2D pose data provided in the dataset as the ground truth. For the pose regression network, the ground truths are the corresponding 3D poses for each image.

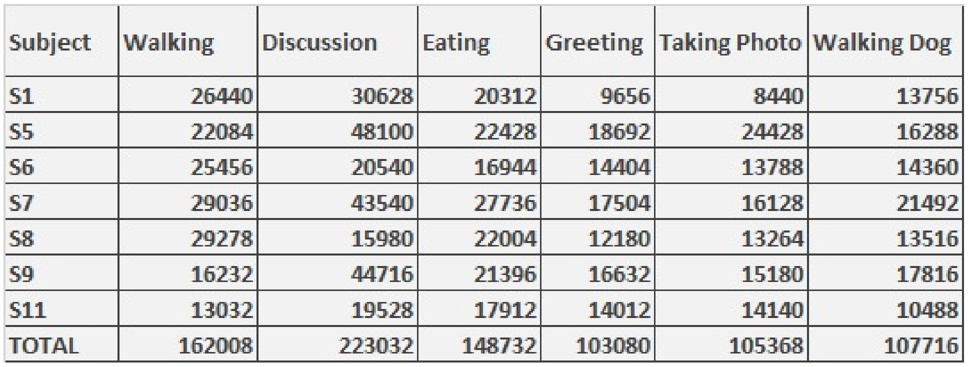
We use a subsection of this dataset, namely the ‘Walking’ action. Also, we use only seven subjects out of the eleven available subjects in the dataset, as the ground truth for the other four are held by the Human36M dataset creators for testing-only. We train the network using the data from six subjects and test our results on the seventh subject. The number of images belonging to each subject for ‘Walking’ as well as some other actions are shown in figure 1(a). As shown in the figure, we use subjects S1, S5, S6, S7, S8 and S9 for training the network which amounts to approximately 0.15 million images and we test our network on S11 which has 13,032 images. We plan to incorporate the other five actions shown in the figure, for which we have already pre-processed the data, in our future work.

The RGB images are frames of the recorded videos from which we crop the human bounding-box and convert to a grayscale image. Also, we us the background subtraction mask data per image provided in the dataset to get rid of the background part of the image. This leaves us with images of varying dimensions, which are then normalized to a fixed size of 128 X 128. We maintain the aspect ratio of each image by padding zero-value pixels along the shorter dimension and ensure that the subject is placed at the center of the image. An image before and after the pre-processing step is shown in figure 1(b) and 1(c) respectively. For the 2D and 3D poses, the dataset originally provides pose data for 32 body joints, out of which we use only 17 main joints, leaving aside unimportant joints like fingers, toes, etc. We use the following 17 joint ids out of the 32 available :

$$J_{(17)} = [1\ 2\ 3\ 4\ 7\ 8\ 9\ 13\ 14\ 15\ 16\ 18\ 19\ 20\ 26\ 27\ 28] \tag{1}$$

And, instead of using absolute joint positions provided in the pose data, we use joint positions relative to the root location, by subtracting from each joint position, the position of the pelvis joint

[1]/cvlabdata1/home/btekin/tracking/spatiotemporal_pose_ estimation/pose_code/Human36m/semester_project/MAINFINAL
[2]https://github.com/arj7192/HumanPose3D

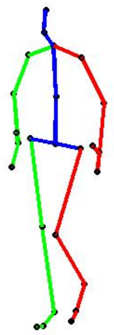


Subject	Walking	Discussion	Eating	Greeting	Taking Photo	Walking Dog
S1	26440	30628	20312	9656	8440	13756
S5	22084	48100	22428	18692	24428	16288
S6	25456	20540	16944	14404	13788	14360
S7	29036	43540	27736	17504	16128	21492
S8	29278	15980	22004	12180	13264	13516
S9	16232	44716	21396	16632	15180	17816
S11	13032	19528	17912	14012	14140	10488
TOTAL	162008	223032	148732	103080	105368	107716

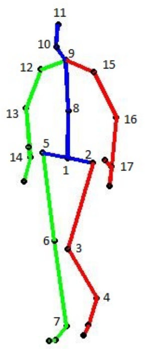
(a) Number of images for each of the seven subjects from Human36M

(b) Original sample image of size 478 X 197 X 3 of subject S1 from 'Walking' action

(c) Pre-processed 128 X 128 grayscale image



(d) 3D pose sample visualization



(e) 3D pose marked with the 17 joints

Figure 1: Human36M dataset

(joint id = 1). The 3D pose ground truth for the regression network is a vector of size 51 X 1, for each 128 X 128 grayscale image as an input to the network. A sample 3D pose visualization corresponding to an image is shown in figure 1(d). Figure 1(e) shows the same 3D pose image marked with the 17 consequential joint ids that we use in our network. Note in the figure that the peripheral joints like fingers and toes are not marked among the 17 joints.

4 Deep network for pose estimation

Inspired by [1], the deep framework consists of two sub-networks : a joint point detection network and a pose regression network. The inputs for both these networks are the bounding box images containing human subjects. The regression network learns the 3D joint positions, whereas the detection network uses a sliding window approach for each of the joints and performs classification task of whether a body joint is contained in a sliding window or not. Initially we began with working only with the pose regression network by random initialization. Upon not achieving good results, and for getting a reasonable initialization of network weights, we started pre-training the regression network with the help of the detection network, which finally gave us promising results.

4.1 Pose Regression

If J_i = ($J_{i,x}$, $J_{i,y}$, $J_{i,z}$) is the position of the i^{th} joint provided in the 3D pose data and if P(i) refers to the pelvis joint, then we use the following relative joint positions, R_i = J_i - $J_{P(i)}$. We use relative joint positions because the absolute positions convey less meaning by themselves and would be harder to train with. The regressor network is trained by minimizing the squared difference between the prediction and the ground truth position,

$$E_r(R_i, \hat{R}_i) = \|R_i - \hat{R}_i\|_2^2 \tag{2}$$

where R_i and $\hat{R}_i$ are the ground truth and predicted positions respectively for i^{th} joint. In the present state, we have used an identity activation function for the last regression layer of the pose regression network. For adding to the complexity of the network, we plan further to use also a non-linear activation for the last regression layer as done in [1]. For the rest of the hidden fully-connected layers, we use the tanh activation.

With a kernel configuration of [32 64 64] derived from [1] and a learning rate of 0.001, this network takes 2 to 3 days to train for 1 epoch on the CV lab CPU server [12]. As an alternative, we also use a smaller kernel configuration of [5 10 15] to speed up the learning process. With the additional optimizations like momentum and weight decay, it requires more than 30 to 40 epochs for the regression network to reach an optimum, whereas with the normal gradient descent procedure, the network seems to saturate within 4 to 5 epochs.

4.2 Joint point detection

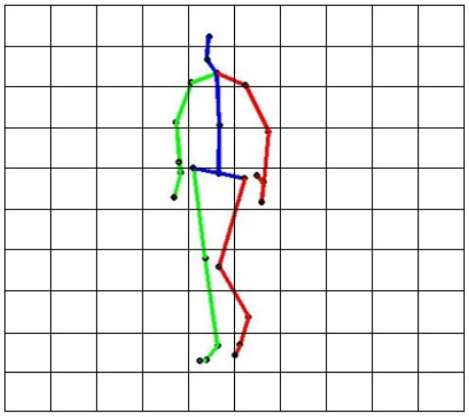
For each image i, and for each joint j, we generate a 10 X 10 2D grid where we have 13 X 13 size windows across the grid. This grid is overlaid on the 128 X 128 image space, where we have approximately 100 such windows. We transform the 2D pose data corresponding to image i, with the same re-sizing parameters used for transforming image i from its original size to a 128 X 128 image. For the 100 windows generated, we have

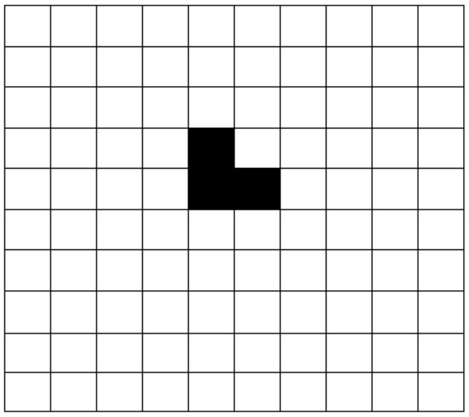
$$h_{j,w} = \begin{cases} 1, & \text{if } J_j \text{ is inside window w,} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

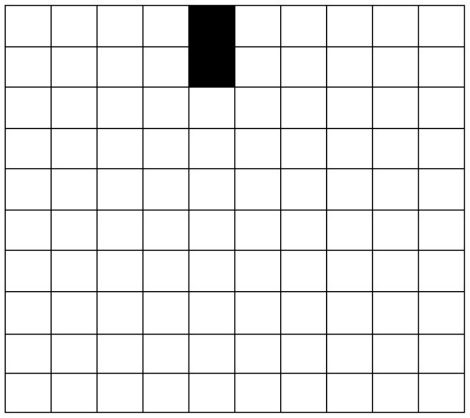
where J_j is the transformed 2D position for joint j, and $h_{j,w}$ is an indicator variable. The goal of this network is to predict this indicator variable for each of the 17 joints for each of the image provided in the dataset. This network has, therefore a 1700 X 1 size binary vector as the ground truth. And the network is trained by minimizing cross entropy between the ground truth label $h_{j,w}$ and predicted label $\hat{h}_{j,w}$,

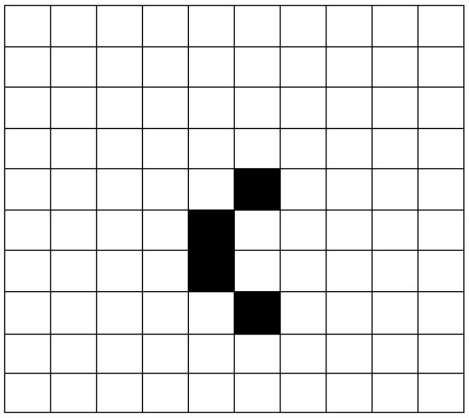
$$E_d(h_{j,w}, \hat{h}_{j,w}) = -h_{j,w} log(\hat{h}_{j,w}) - (1 - h_{j,w}) log(1 - \hat{h}_{j,w}) \tag{4}$$

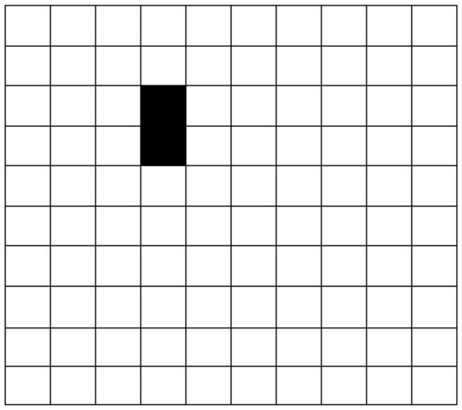
Visualization of the sliding window approach for detection is shown for a few joints in fig 2. The procedure shown in the figures 2(b), 2(c), 2(d) and 2(e) for the four joints is performed for all 17


(a) A 10 X 10 grid overlaid on the 128 X 128 2D pose space


(b) Sliding window detection for the pelvis joint (joint id = 1)


(c) Sliding window detection for the head (joint id = 11)


(d) Sliding window detection for the left knee joint (joint id = 3)


(e) Sliding window detection for the right elbow joint (joint id = 13)

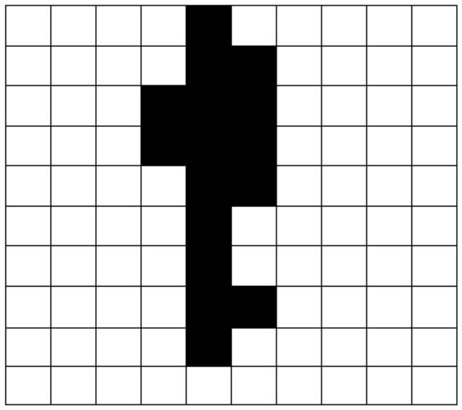

(f) Sliding window detection combined for all the 17 joints

Figure 2: Joint point detection task schematic

joints for each training sample. Notice that we have more than one window detected for a particular joint. This is because for each joint point, we consider not only its position but also the edges connecting this joint point to its immediate neighbours. If more than 30% of the edge lies in a window, that window is also considered as detected. This particular algorithm is inspired from [3].

For this network, we use the tanh activation for all the hidden fully connected layers including the last classification layer. This network takes approximately the same amount of time per epoch as the regression network for the same set of hyper-parameters. We train this network just once for upto 50 epochs with a convergence criterion. Thereafter, we load the saved weights of this network for training the regression network each time instead of a random initialization.

4.3 Full network

The network has 9 layers : 3 convolutional layers shared by both the sub-networks, 3 fully connected layers for regression, and 3 fully connected layers for detection network.

We first train exclusively on the detection network using the following cost function :

$$\phi_M = \frac{1}{2}\sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{l=1}^{L} E_d(h_{i,l}^{(t)}, \hat{h}_{i,l}^{(t)}) \tag{5}$$

and then train the regression network using the learned weights of the first three convolutional layers from the detection network, using the following cost function :

$$\phi_M = \frac{1}{2}\sum_{t=1}^{T}\sum_{i=1}^{N} E_r(R_i^{(t)}, \hat{R}_i^{(t)}) \tag{6}$$

where t is the index of training sample, N is number of joints and T is number of training samples. A schematic of the network architecture derived from [1] is shown in figure 3. In the figure, N represents the number of joints and N=17 in our case. For the first and second convolutional layers conv1 and conv2, we use filters of size 5 X 5 and for the layer conv3, we use filters of size 6 X 6. For both of the sub-networks, we use back-propagation to update the weights during training.

The first convolutional layer conv1 filters the 128 X 128 input image with 32 kernels of size 5 X 5. The second convolutional layer conv2 takes as input the max-pooled (with a pooling size of 2 X 2) output of conv1 and filters it with 64 kernels of size 5 X 5. Similarly, the third convolutional layer conv3 has 64 kernels of size 6 X 6 to filter the pooled output of conv2. The choice of this particular architecture is primarily inspired by [1]. Effectively, in the first phase of the learning process, the link between layers pool3 and fcr1 is effectively cut-off and only the detection network is running. And in the second phase, we cut-off the connection between layers pool3 and fcd1 and and connect pool3 to fcr1 to run the regression network.

4.4 Learning details

In the first session of our experiments, we trained only the regression network with batch size of 1 image and learning rate of 0.001. After getting unconvincing results even after changing the batch size to 128 images, we switched to using the detection network as well for pre-training as suggested in [1]. Furthermore, we incorporated more optimizations to get good results and in the present state of our work, we train our model using stochastic gradient descent with a batch size of 128 examples, learning rate of 0.001, momentum of 0.9 and weight decay of 0.0005. These parameter values are mainly inspired by [2]. The update rule for weight w is

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \langle \frac{\partial L}{\partial w} \mid_{w_i} \rangle_{D_I} \tag{7}$$

$$w_{i+1} := w_i + v_{i+1} \tag{8}$$

where i is the iteration index, v is the momentum parameter, ϵ is the learning rate, and $\langle \frac{\partial L}{\partial w} \mid_{w_i} \rangle_{D_I}$ is the average over the i^{th} batch D_i of the derivative of the cost function with respect to w, evaluated at w_i

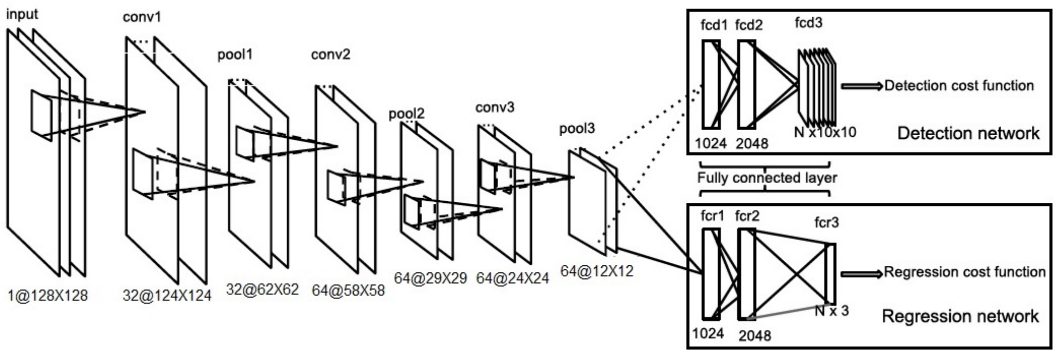


Figure 3: Deep Convolutional Neural Network Architecture

Number of Convolutional Layers	**3**
Number of Fully Connected layers	**3(Detection) + 3(Regression)**
Input size	**128 x 128**
Output layer for Detection	**1700 nodes**
Output layer for Regression	**51 nodes**
Kernel configuration(s) in 3 Convolutional layers	**[5 10 15] and [32 64 64] (both used)**
Learning Rate	**0.001**
Weight Decay	**0.0005**
Momentum	**0.9**
Batch Size	**128**
Activation Function	**tanh (except for last regression layer)**
Weight Initialization for Detection	**random**
Weight Initialization for Regression	**Learned weights from detection network**

(a) Network Parameters

First convolutional layer	**81920**
Second convolutional layer	**47610**
Third convolutional layer	**12615**
First Fully Connected Layer	**1024**
Second Fully Connected layer	**2048**
Output layer	**1700 for detection, 51 for Regression**

(b) Number of nodes

First Conv Layer	**5x5 and (2,2)**
Second Conv Layer	**5x5 and (2,2)**
Third Conv Layer	**6x6 and (2,2)**

(c) Filters and (max)pool size

Figure 4: Details of learning

For the detection network, we initialize the network with random weights, and for the regression network, we use learned weights from the detection network for initialization. We maintain a constant learning rate throughout the learning procedure. We trained the detection network for 50 epochs and we trained the regression network for 100 epochs, with the dataset of 0.15 million images and kernel size of [5 10 15], which takes approximately five to six days on the CV lab CPU server [12]. We do not use yet any specialized over-fitting removal techniques such as dropouts or rectified linear units. The network parameters that we used finally are summarized in figure 4.

5 Results

For the detection network which we use to pre-train the regression network, we obtain a very low train error of around 0.05 (cross-entropy error) by the end of 50 epochs. The learning curve for the detection network is shown in figure 5(a). For the regression network, we evaluate the pose predictions using the mean per joint position error (MPJPE).

$$\text{MPJPE} = \frac{1}{T}\frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N}\|(J_i^{(t)} - J_{root}^{(t)}) - (\hat{J}_i^{(t)} - \hat{J}_{root}^{(t)})\|_2. \tag{9}$$

We finally used [5 10 15] as our kernel configuration to be able to train the regression network for 100 epochs. For our dataset that involves only the ‘Walking’ action, we obtain the best test error of 80.42 and corresponding train and validation errors of approximately 36.23 and 40.7 respectively in terms of MPJPE. The learning curve for the regression network for 100 epochs is shown in figure 5(b). From the learning curve, we observe that right after epoch 7, the train and validation errors, along with the test error, start declining until epoch 100. We used weight decay, due to which the network goes on learning progressively for greater number of epochs. We observe that the test error almost saturates by epoch 100.

Qualitatively, we visualize the predicted 3D poses for certain test samples and compare it with the ground truth. We use the visualization code provided along with the Human36M dataset [3]. We find that the network is able to predict a good approximation for the ground truth 3D pose although the predictions are not as smoothly changing with increasing number of frames as the ground truth. A few predicted 3D poses are shown along with the corresponding ground truth in figure 6.

On visualization of our results, we observe a piece-wise constancy in the prediction of 3D poses compared to the smoothly varying ground truth. We found that the network predicts almost a constant pose for k consecutive neighboring ground truth poses (k=12 mostly in our case), i.e. the prediction pose changes after every k frames of ground truth 3D pose. The network seems to be trying to approximate a single pose prediction for consecutively multiple ground truth 3D poses. The video with visualization results comparing the ground truth pose and the predicted pose are available for validation [4] and test data [5].

6 Conclusion

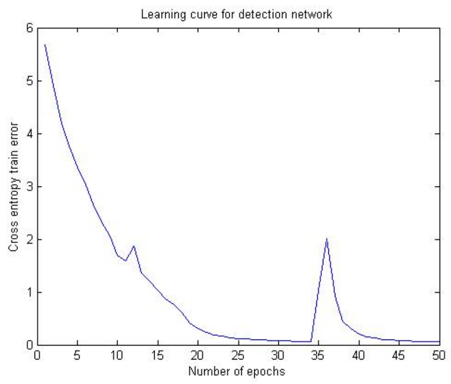
In this project, we use a deep convolutional neural network for estimating 3D human pose from grayscale images. We use the Human36M dataset for training the network. The pre-processing step of the images involve RGB to grayscale conversion, cropping of the human bounding box, compression, padding with the subject at the center. We also pre-process the 2D and 3D pose data accordingly. We pre-train the regression task with the detection task, which helps in regularization. The detection network is run once for 50 epochs and then the learned weights for layers conv1, conv2 and conv3 are used for initialization in the regression network.

We use stochastic gradient descent for training the regression network along with optimizations like momentum and weight decay. With a kernel configuration of [5 10 15] for the three convolutional layers, upon training for 100 epochs, the regression network reaches its optimum, and the train and test errors saturate at around 36 and 80 (MJPPE) respectively. We use MPJPE as the error metric

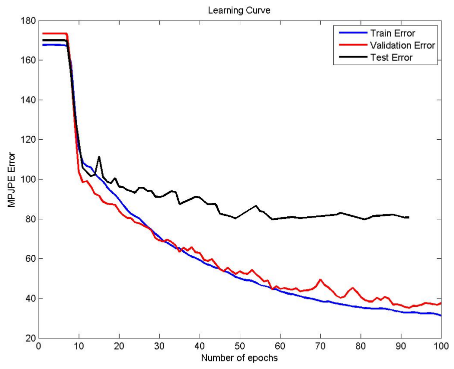
[3]http://vision.imar.ro/human3.6m/code-v1.1.zip

[4]https://drive.switch.ch/public.php?service=files&t=d14ace048e9613e3916856e03d8fe4ac

[5]https://drive.switch.ch/public.php?service=files&t=a5a28fb4cea80f480ae991922a837f7b

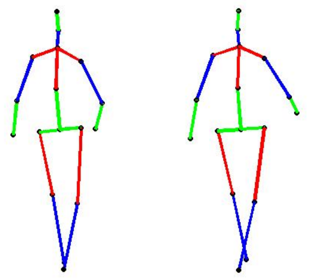


(a) Joint point detection network

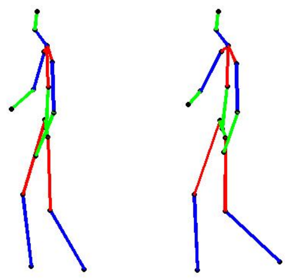


(b) 3D pose regression network

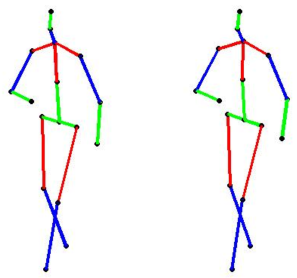
Figure 5: Learning curves for detection and regression networks



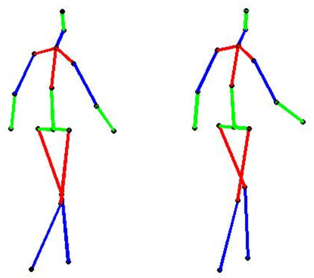
(a) Test result 1 : prediction(left) and ground truth (right)



(b) Test result 2 : prediction(left) and ground truth (right)



(c) Test result 3 : prediction(left) and ground truth (right)



(d) Test result 4 : prediction(left) and ground truth (right)

Figure 6: Sample test results for 3D pose regression

for evaluation and obtain the best test error of 80.43. Upon visualization of the predicted 3D poses compared to the ground truth, we notice that we obtain a promising prediction 3D pose for all the test samples. However, the predictions are changing at a lower frequency compared to the ground truth 3D pose samples. Future work will be on alleviating this piecewise constancy and predicting smoothly varying poses for each of the test samples. We also plan to include five more actions other than 'Walking' to augment our data and to have a better generalization.

Acknowledgments

I would like to acknowledge my classmates Arun and Isinsu for helping me in this project. And most importantly, I would like to thank my supervisor Bugra for providing me with the required resources, for guiding me and for clearing my doubts throughout the course of this project. His supervision and guidance was really helpful and necessary for understanding and dealing with the complexities of the powerful deep neural networks.

References

1. Li, Sijin, and Antoni B. Chan. ”3D Human Pose Estimation from Monocular Images with Deep Convolutional Neural Network.”
2. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. ”Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012.
3. Heterogeneous Multi-task Learning for Human Pose Estimation with Deep Convolutional Neural Network
4. Moeslund, Thomas B., Adrian Hilton, and Volker Krger. ”A survey of advances in vision-based human motion capture and analysis.” Computer vision and image understanding 104.2 (2006): 90-126.
5. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. Theano: A CPU and GPU Math Expression Compiler. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin
6. Toshev, Alexander, and Christian Szegedy. ”Deeppose: Human pose estimation via deep neural networks.” Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014.
7. Shotton, Jamie, et al. ”Real-time human pose recognition in parts from single depth images.” Communications of the ACM 56.1 (2013): 116-124.
8. Chapter 11, *Introduction to Statistical Learning* by G. James, D. Witten, T. Hastie and R.Tibshirani
9. Ionescu, Catalin, et al. ”Human3. 6M: Large scale datasets and predictive methods for 3D human sensing in natural environments.” Pattern Analysis and Machine Intelligence, IEEE Transactions on 36.7 (2014): 1325-1339.
10. F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. Theano: new features and speed improvements. NIPS 2012 deep learning workshop.
11. Catalin Ionescu, Fuxin Li and Cristian Sminchisescu, Latent Structured Models for Human Pose Estimation, International Conference on Computer Vision, 2011
12. http://wiki.epfl.ch/cvlab-it/resources/computing