Department of Computer Science

Feature descriptors for gait analysis from depth sensors

Ben James Crabbe

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2015 | CSMSC-15

Declaration:

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Ben James Crabbe, September 2015

Contents

1 Introduction

Gait analysis plays an important part in the treatment and assessment of a number of medical conditions. At present gait analysis is usually performed through a combination of visual assessment by an experienced physiotherapist, automated methods such as marker based motion capture, pressure sensitive walkways or accelerometers. It generally involves travelling to a gait assessment laboratory and this can be an issue for patients who have difficulty walking.

This problem, and a range of other healthcare challenges, are being tackled through research and development by the SPHERE (a Sensor Platform for Healthcare in a Residential Environment) group in Bristol. An automatic, in home, gait analysis pipeline has been designed [**?**, **?**] which assesses the quality of a subjects movement using data captured by the human pose estimation (skeleton/joint tracking) systems implemented by RGB-D cameras such as the Microsoft Kinect.

The system uses data driven, machine learning methods to learn both a low dimensional representation of pose [1] and models of normal motion from which it quantifies the quality of movement. Because of this, it can be applied naturally to other types of movement quality assessment such as sports movement optimisation or physiotherapy exercise coaching. The system has been applied to a sitting-standing motion, to punching motions in boxing [**?**] and to people walking up stairs [**?**, **?**].

One issue currently limiting the effectiveness of this system is the fragility of the skeleton tracking software. This software was designed for controlling entertainment/gaming systems with the user viewed frontally, within a range of 1-4m and at a pitch angle near 0°. Outside of these conditions skeletons become noisy and unreliable. Typically only a small fraction of data recorded from a camera fixed ∼1.5m above the top of stairs is fit for use with the system. Increasing the fraction of usable data requires more intrusive camera placement which is to

[1] The points in this space are referred to as the pose vector, and the Kinect skeleton data as the body configuration or joint position vector.

be avoided. The skeleton tracker also performs poorly when props are involved, for example grasping a banister or a ball often leads to erroneous joint positions for that arm. It also struggles to accurately record sitting/standing motions which is a key motion hoping to be analysed in SPHERE homes.

The aim of this project was to develop a system for determining the low dimensional pose representation used in [**?**] directly from RGB-D footage, with the additional aims that the system should be naturally applicable to a range of motions and able to maintain reasonable accuracy at a range of viewing angles. The methodology we found most suited to this task was a convolutional neural network (CNN). CNN's are a supervised learning method for extracting features, e.g. the pose vector, from images. After training the network is able to generalise to unseen data, producing an output inferred from the examples it has seen. This method effectively learns the feature representations that produce the most accurate regression. The advantage of this over traditional hand crafted feature extraction is that when applied to new motions the type of features extracted will be re-tuned to better capture the pose vector from the data of the new motion.

One restriction of CNN's is that they require a large amount of data to be trained effectively. In this project, as in [**?**, **?**] we have focused on an analysis of the stair ascent motion as this was the only existing dataset of the required size. We use the joint position data captured by the Kinect to produce the ground truth pose vectors used to train the CNN. The Kinect skeletons are not ideal ground truth; by using them we are restricted to only motions which the Kinect is able to capture. Also, their errors and imprecision produce an inconsistent mapping between similar images and labelled pose vectors which can confuse the network. Despite this we find that in a majority of cases the CNN is able to match the Kinect measurement to a high degree, and in some cases predict poses which correspond better to what is seen in the images than was produced by the Kinect. The accuracy of the CNN is measured by the distance (euclidean/L2 norm) between its predicted point and the ground truth. This is used during training as a objective/loss function and in the analysis of the final performance. The effect of using the CNN's pose measurements for the movement quality analysis of [**?**, **?**] is also examined. Comparing it to that of the ground truth we find that in some cases the network predictions are actually more adept at producing the correct abnormality than the Kinect skeletons.

In Section **??** we present an introduction to depth imaging, human pose estimation and convolutional neural networks. In Section **??** we detail the steps used to achieve our results. In Section **??** we present and analyse our results. In Section **??** we evaluate the suitabilty of our chosen method and present suggested directions for future work.

2 Background and Related Work

2.1 Depth Imaging

In depth images each pixel value represents the distance of that point from the camera. Depth images are unaffected by changes in lighting or human appearance. They provide a 3D map of the scene, making background-foreground separation far easier. These features can often simplify computer vision tasks, particularly human pose estimation [**?**].

There are three main technologies used to produce depth images: Time of flight (ToF) cameras, Stereo imaging cameras and Structured light cameras. It was not until the last 5 years that affordable good quality RGB-D sensors came on the market and since then there has been an explosion in their use in the computer vision community [**?**].

The data in the SPHERE staircase dataset was captured using a Asus Xtion Pro Live which uses the structured light technology developed by Primesense (same as the Microsoft Kinect). It consists of an infrared laser emitter and an infrared camera, which together make up the depth sensor, and a RGB camera. An infrared laser is passed through a diffraction grating to produce a known pattern of dots that is projected onto the scene then reflected back and captured by the infrared camera. The measured pattern is compared to a reference pattern produced at a known distance of reflection, which has been stored during the calibration process. The surface of reflection being farther or nearer than the reference surface produces a shift in the pattern which is used to determine the depth value [**?**, **?**] as shown in figure **??**.

2.1.1 Sensor Performance

Most of the studies reported below have focused on the Microsoft Kinect however both sensors contain the same depth sensing system and have been shown to perform equivalently when compared [**?**], hence we report the findings based off of the Kinect performance.

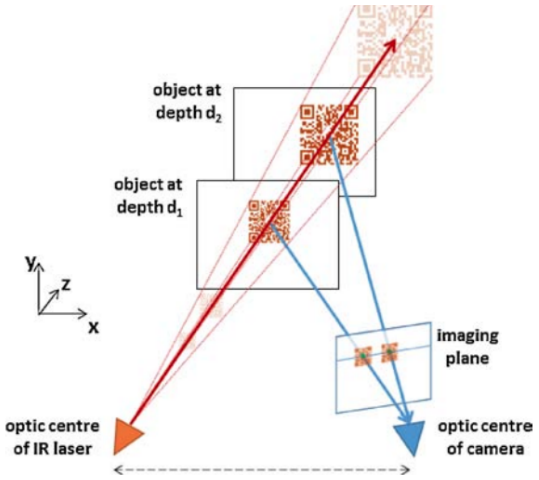


Figure 1: The process by which depth is computed from triangulation of structured light. From [?]

The range of the depth sensor is 0.8-3.5m with increasingly noisy or incomplete readings up to 8m. It has a 43° vertical by 57° horizontal field of view [?].

Stoyanov et al. [?] compare the performance of the Kinect with that of two other ToF depth imaging cameras (SwissRanger SR-4000 and Fotonic B70 ToF) assessing them against a ground truth of expensive and low fps laser depth scanner measurements. They find that within a range of 3.5m the Kinect outperforms the ToF sensors and is comparable to the laser scanner, outside of this range the accuracy falls considerably.

Both Khoshelham & Elberink [?] and Smisek et al. [?] have measured this effect experimentally comparing Kinect measurements with those from high performance laser scanners. They find that temporally fluctuating noise in the depth measurements increases quadratically with distance from the sensor so the depth precision decreases from around 0.5cm at 1m to 7cm at 7m. Nguyen et al. shows that noise increases linearly with lateral distance, and is greatly increased on surfaces at greater than 70° angles [?]. This last effect can lead to increased levels of noise around edges of humans.

As well as noise the structured light sensors often return 'unknown' depth value pixels, known as holes, when the infrared receiver cannot read the reflected pattern properly. This can occur around the sides of foreground objects due to the slightly different viewing angles between the projector and camera as in regions 2 and 3 of figure **??**, or when certain surface

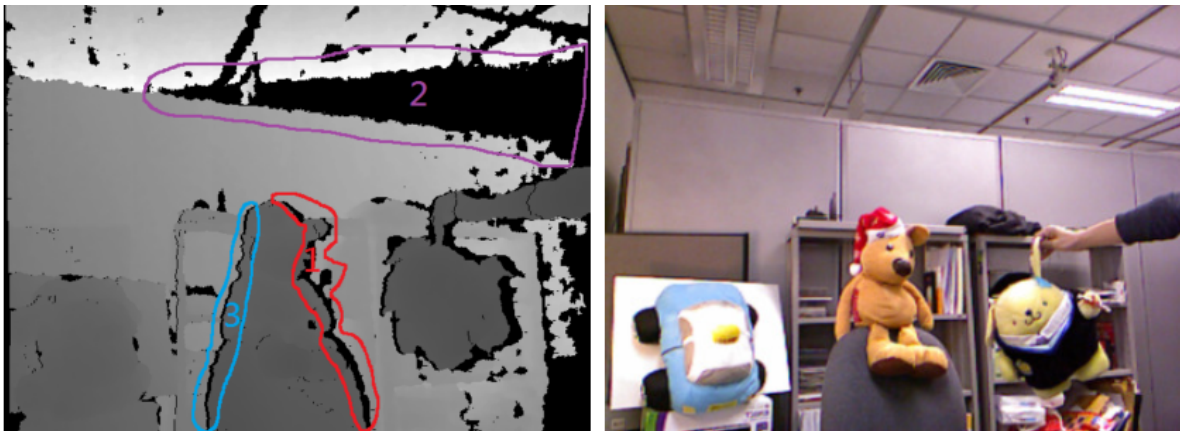


Figure 2: Shows the holes in structured light depth data due to the different perspectives of IR projector and senor (regions 1 and 3) and due to the surface of reflection being roughly 5m away and at a large angle(region 2) From [**?**]

materials, such as human hair, interfere with the infrared pattern's reflection as in region 4 in figure **??**.

It should be noted that each of these studies mentioned above [**?**, **?**, **?**] fail to consider the environmental factors in the quality of the measurement. Fiedler & Muller [**?**] show that air draft can cause changes of the depth values up to 21mm at a distance of 1.5m, and temperature variations cause changes up to 1.88mm per 1° C. They also find a temperature dependant drift in the position of objects captured by the RGB camera.

2.2 Human Pose Estimation

Human pose estimation (HPE) is generally considered as the task of measuring in 2D or 3D the joint positions of the human body. It is one of the most researched problems in computer vision because of its difficulty and due to its use in a variety of applications such as video surveillance, humancomputer interaction, digital entertainment, sport science and medical applications.

This is a difficult task for a number of reasons. Firstly the human body has around 20 degrees of freedom [**?**], producing a huge space of possible body configurations, many of which will cause some joints to be occluded when viewed from a single camera. Additional difficulties arise from the variety in human appearance and clothing, and from left right ambiguities.

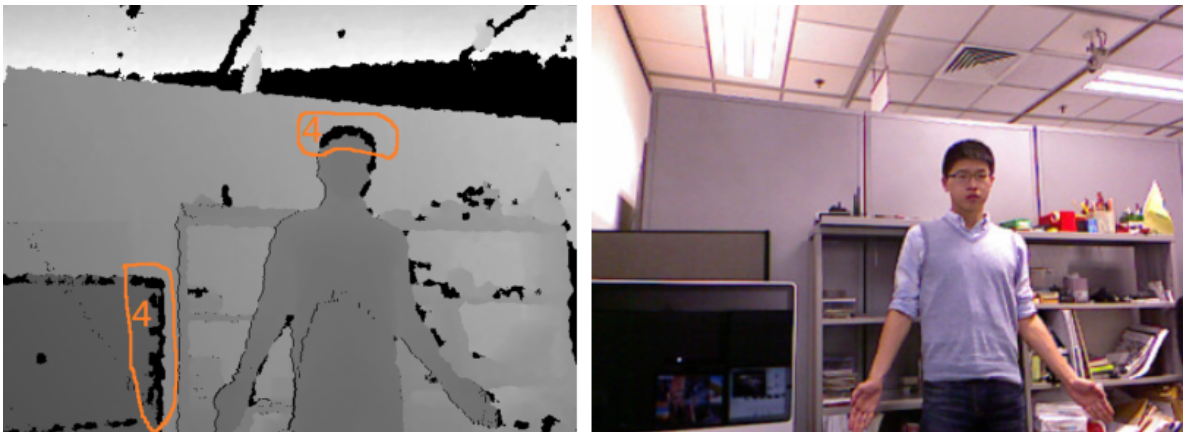


Figure 3: Shows the holes in depth data due abnormal reflections from certain glossy surfaces like the TV monitor and the subjects hair. From [**?**]

Traditional motion capture methods (MoCap) rely on markers attached to the subject and multiple cameras to overcome these issues. Whilst such systems can provide highly accurate pose data, their use is restricted to controlled environments using expensive and calibrated recording equipment which renders them unsuitable in many applications.

Monocular visual pose estimation methods (reviewed in [**?**, **?**, **?**, **?**, **?**]) are generally divided into two approaches (e.g. by [**?**]); model based (or generative) and model-free (or discriminative) approaches. Model based approaches use prior knowledge of human shape and kinematics such as fixed limb lengths and defined joint angle limits to cast the image to pose transformation as a nonlinear optimisation problem or probabilistically in terms of a likelihood function, i.e. given this image data (and sometimes previous frames pose knowledge) what is the most likely valid pose. Model-free approaches instead learn a direct mapping from image data to pose, generally requiring learning/example based methods to achieve this. Some 'hybrid' approaches combine the two using model-free methods as an initialiser to model based methods.

2.2.1 Human Pose Estimation Using Dimensionality Reduction

With both of the above approaches there are significant issues posed by the high dimensionality of pose data. In model based approaches likelihood functions, which are usually multi-modal and non-Gaussian, require a randomised search [**?**]. Such searches in 20 dimensions are computationally expensive and often lead to super real time frame rates [**?**]. In model free

approaches training data must account for the highly non-linear mapping between image and pose, which means that the pose space must be densely sampled in the training set. Densely sampling a 20 dimensional space, even just the parts that correspond to valid human motion, whilst also modelling all the invariant aspects such as body shapes, viewing angle etc requires an inordinate amount of data [**?**, **?**].

Although the full pose space is very large and high dimensional it has been shown e.g. in [**?**, **?**] that when considering only movements in a well defined activity such as walking, then the pose data can be well represented by a low dimensional latent manifold. In a work closely related to our own Elgammal et al. [**?**] use Local Linear Embedding (LLE), a dimensionality reduction method, to generate a 1D manifold representation (embedded within a 3D space), shown in figure **??**, of a walking motion from single sequences of silhouette images. They use a Generalised Radial Basis Function interpolation framework [**?**] (a form of neural network) to learn two mappings, one from the manifold to the silhouette image space and another from the manifold to full 3D joint positions. They then invert these mappings to extract points on the manifold from silhouettes and 3D joint positions from the points on the manifold. In contrast, our work builds the manifold representation from 3D joint position data, this has the benefit of the manifold representation not being tied to subject's visual appearance. This allows it to generalise naturally to multiple subjects, which is not the case in [**?**] (although they do introduce an solution for this problem in [**?**]). It is also unlikely that this method could be used to capture abnormality in gait since defining an image to manifold transformation explicitly from the inverse constrains all input images to the poses contained in the original data. Elgammal et al. argue that learning a smooth mapping from examples is an ill-posed problem unless the mapping is constrained since the mapping will be undefined in other parts of the space. We address this issue in section **??**.

Brand [**?**], also inferred 3D pose from silhouettes using an intermediate manifold representation. He uses a maximum a posteriori estimation for mapping between the image and manifold space. This uses information across the whole input sequence to find the most likely and consistent solutions in order resolve the ambiguities in the many to many silhouette to pose mapping. A solution of this form is unacceptable in our case as one of the key features of the SPHERE system is online measurement.

Similarly Urtasan et al. [**?**] used Scaled Gaussian Process Latent Variable Models (SGPLVM)

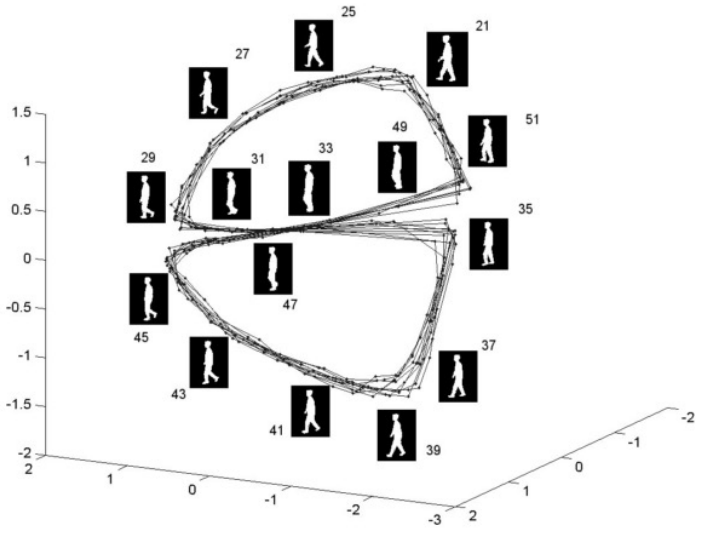


Figure 4: The 1D gait manifold produced from LLE dimensionality reduction on images of shilouttes used by Elgammal et al. to simplify joint tracking. From [**?**].

[**?**] on single training sequences (walking and a golf swing) labelled with 2D joint positions. These models build a low dimensional manifold, and simultaneously, a continuous mapping between this and the 2D joint positions. They use the low dimensional representation to facilitate efficient maximum a posteriori based tracking in this space. Again, this is only suitable as an offline solution.

Tangkuampien and Suter [**?**] used Kernel Principle Component Analysis (KPCA) to learn low dimensional representations of 3D joint positions, and separately using the same method, a low dimensional representation of silhouette images. They then learn a mapping between these spaces using Locally Linear Embedding (LLE) reconstruction. This has some similarity with using a CNN, where the image is transformed in to progressively more concise feature representations, before the regression takes place. The disadvantage of using LLE to perform the final transformation is that it is restricted to within the manifold space contained in pose training data.

Rosales et al [**?**, **?**] used 3D joint position data from a MoCap system to render synthetic training data from multiple angles. Hu moments were used to extract visual features from these (and real) images. Unsupervised learning was used to cluster 3D joint position data into areas of similar pose and a neural network was trained separately on each cluster to

learn the mapping from visual features to pose. Using the developments in training of deep neural networks since this work we show that it is feasible to have a single CNN learn both the features best suited and the mapping to all poses. This removes the need for clustering and separate networks leading to a simpler, easily adaptable solution. Although their use of MoCap data as opposed to Kinect Skeletons for ground truth is a change we could expect to improve performance in our system. Similarly rendering synthetic training data from multiple angles would be a smart way of improving our viewing angle tolerance.

2.2.2 Human Pose Estimation From Depth Images

Challenging aspects of RGB HPE such as the variability in human appearance and scene lighting were greatly simplified with the advent of low cost commodity depth sensors. RGB-D also provides richer data for inferring 3D structure, allowing human poses which could appear identical when projected onto a 2D image plane to be distinguished. Full body HPE methods from single depth images are reviewed in [**?**]. With the Kinect/Xtion sensor and software packages (Kinect SDK or alternatively the open source OpenNI) low cost, flexible and reasonably accurate HPE is now available and has been employed in a huge variety of scientific applications [**?**, **?**].

The Kinect SDK and OpenNI skeleton trackers apply some inter-frame tracking algorithms to the single frame pose measurements of [**?**]. In this work Shotton et al. leveraged a large MoCap 3D joint position dataset which they re-targeted onto a variety of synthetic body models before rendering as if captured from a Kinect, simulating sensor noise, camera pose, and crop position. Producing synthetic depth images data is far simpler than in RGB since depth is far more invariant to subject clothing and appearance changes. Using these generated depth images and a ground truth labelling of each pixel as one of 31 body parts they trained a randomised regression forest to perform this body part classification at each pixel using simple and computationally efficient pixel wise depth comparison features

$$f(\boldsymbol{u} \mid \boldsymbol{\phi}) = z(\boldsymbol{u} + \frac{\boldsymbol{\delta}_1}{z(\boldsymbol{u})}) - z(\boldsymbol{u} + \frac{\boldsymbol{\delta}_2}{z(\boldsymbol{u})}) \tag{1}$$

where $\boldsymbol{u}$ is a pixel location, $\boldsymbol{\phi}$ is two randomly generated 2D offsets $\boldsymbol{\delta}_1$, $\boldsymbol{\delta}_2$, $z(x, y)$ is a function which returns the depth at the input location. Examples of these features are shown in figure **??**.

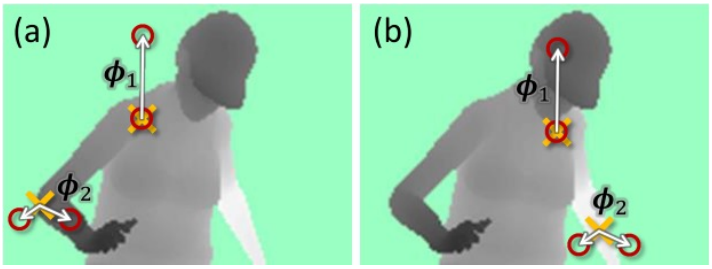


Figure 5: An example the depth comparison features from equation **??** used in [**?**] to perform per pixel body part classification. The yellow crosses indicate the image pixel $\boldsymbol{u}$ being classified. The red circles indicate the offset pixels as defined in equation **??**. They use random forests which combine many such features to give a strong discriminative signal. From [**?**]

They then use these classifications to infer actual joint position through a simple averaging and mean shift procedure. The whole algorithm operates in real time on the computational resources allowed to them on the Xbox gaming consoles GPU. [**?**] adapts this work by allowing pixel classifications of a number of surrounding joints to be used when estimating the joint position, rather than the single corresponding body part pixels as in [**?**]. This is shown to improve the quality of the prediction for occluded joints. A similar use of MoCap joint position data for rendering synthetic images from multiple views is suggested as an ideal way for increasing the view angle and subject invariance of our system.

Another discriminative method is [**?**] where they find geodesic extrema (which are expected to correspond to the feet, hands and head) from Dijkstras algorithm on a graph produced by connecting all depth pixels in the image into a map. These points are identified (as hands feet or head) by applying local shape descriptors around the area.

Other methods e.g. [**?**, **?**, **?**, **?**] have focused on improving temporal smoothness of the measured pose by combining such discriminative methods with model based temporal tracking methods.

In a recent attempt Chan et al [**?**] use 3D point cloud information and propose a viewpoint and shape histogram feature based off these point clouds. This feature is then used to categorise the pose based on the action being performed using an introduced action mixed model. Each action is prescribed its own low dimensional manifold which allows a human pose database containing a limited amount of data to probabilistically infer the full pose.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are biologically inspired supervised learning systems for extracting features from images. Essentially their goal is to learn an approximation of the function

$$\boldsymbol{Y} = F(\boldsymbol{Z}, \boldsymbol{W}) \tag{2}$$

where $\boldsymbol{Z}$ is an input image, the output $\boldsymbol{Y}$ is the inferred pose vector for this image, and $\boldsymbol{W}$ are the trainable parameters of the network.

They consist of a set of filter maps, essentially matrices (or tensors if applying to multiple channels), which are applied repeatedly across the whole of the input. Each application of these filters produces an activation value which is the sum of each filter element multiplied by its corresponding input pixel value plus a shift term known as a bias. The filter is applied iteratively across the whole image, typically with some overlap. This builds up an 'activation map' for that filter, with each layer being comprised of a number of filters. The activations of each filter are stacked together, becoming the new image which is passed onto the next layer. This process is illustrated in figure **??**. This type of layer, known as Convolution layers, are typically followed by a non-linear function which is what enables the CNN to learn non-linear transformations such as the image to pose transformation we require. Although it is possible to stack convolutional layers on top of each other they are often followed by a pooling layer (also called subsampling). The idea of pooling is to reduce the spatial size from the previous layer as seen in figure **??**. Operating on each depth slice individually, i.e. each filters activation, the pooling window moves across the image taking the values of the elements in the input, conglomerating them using some operation, typically taking the max value as seen in the right hand side of figure **??**. CNNs also typically contain one or more fully connected layers at the end. Being fully connected means that rather than having a small filter applied repeatedly across the input, a number of filters of the same dimensions as the input are applied to the whole volume. This then outputs a $1 \times 1 \times K$ volume where K is the number of filters. This is then identical to regular neural networks where each unit in a layer is connected to every unit in the next. In our case, fully connected layers take the final high level feature representation from the rest of the network and perform the final regression to the pose vector.

[2] http://cs.stanford.edu/people/karpathy/convnetjs/

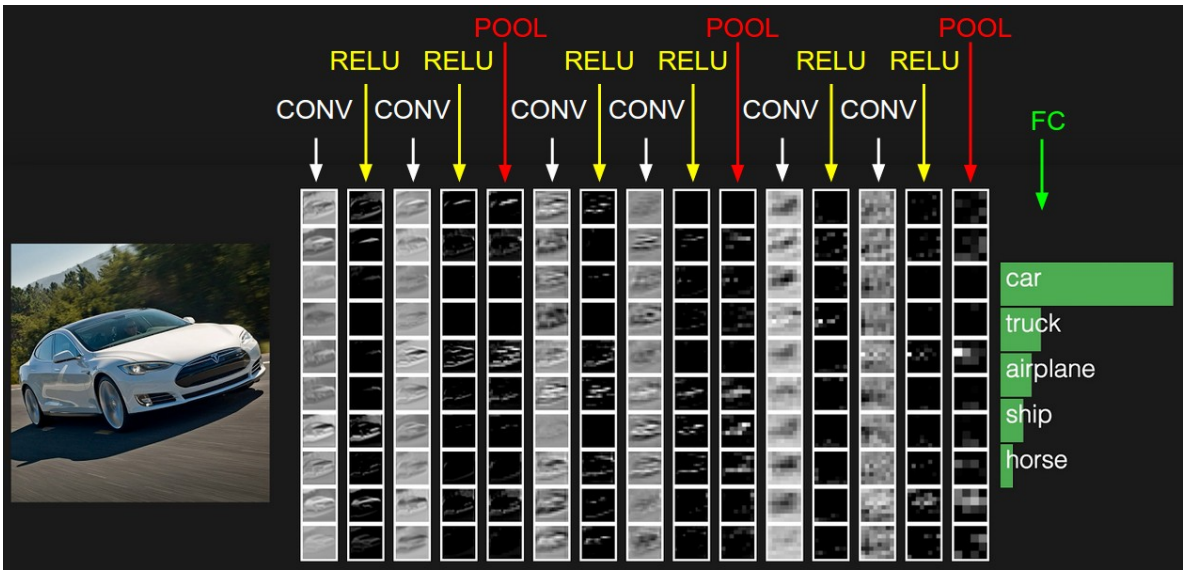


Figure 6: Shows a representation of the activations produced following a number of convolution layers, non-linearities (Rectified Linear Units, or ReLUs, are the function $max(0, x)$ and pooling layers. The network, an online demo from [**?**], is classifying images from the CIFAR-10 dataset using the ConvnetJS library[2].)

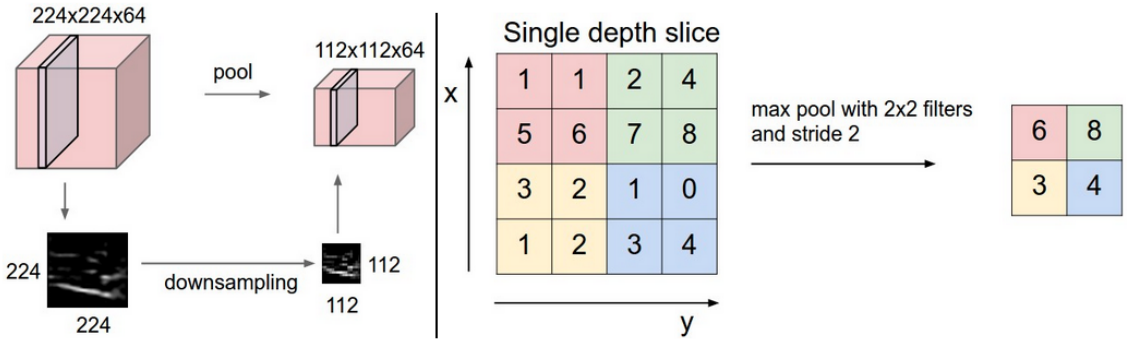


Figure 7: Shows the effect of pooling layers in a CNN. From [**?**]

CNNs are trained using backpropagation and gradient descent. During training an error function is defined which quantifies the difference between the networks output and the desired output. In our case, as is typical of regression tasks, we use the euclidean distance between the two. Using the backpropagation algorithm [**?**] the derivative of this error with respect to each parameter is found. Then the values of each parameter are adjusted a small amount in the direction which reduces the error. In this way, over many training examples, the network converges on a minima in the error surface across the space of all parameter values. A derivation of the backpropagation equations and further technical discussion of CNNs in general can be found in the preceding work on this project [**?**].

An advantage of CNNs over traditional computer vision methods is that rather than prescribing a hand-engineered feature such as the depth disparity feature in [**?**] or the view point and shape feature of [**?**], the network is responsible for learning features itself based off the data. This presents a significant advantage in our application since features which might be good for measuring pose for one type of motion may not be useful for other motions.

Deep CNNs have a proven capacity to learn a huge number of different visual representations. They have achieved great success in the International Large Scale Visual Recognition Challenge (ILSVRC, or ImageNet challenge) which is conducted each year and requires identification and localisation (separately) of 1000 different types of object. The work of [**?**] showed how deep networks could be trained by using ReLU non-linearities in place of the previous staple of tanh or logisitic functions, this over came the vanishing gradient problem that had previously made large network infeasible. Since then CNNs have achieved the best results in each task every year running, with every single participating team using them by 2014 [**?**].

Training deep networks requires a large amount of data. Without this the network may begin to over fit the training data. When this occurs the errors for the network on inputs not in the training set, known as the test set, increases. It can be shown [**?**, **?**] that the difference between the errors on test set and those on the training set is related to both the size of the training set, P, and the capacity, c, of the network i.e.

$$E_{test} - E_{training} \propto \frac{c}{P} \tag{3}$$

The capacity of the network is essentially the number of parameters being trained, which depends on the number of layers, the dimensions of the input images and the number of channels,

the size and number of convolving filters and the way the filters and the pooling are applied. Increasing capacity will also decrease the size of $\boldsymbol{E}_{training}$, hence there the aim in designing the network is to find the architecture which minimises both $\boldsymbol{E}_{training}$ and $\boldsymbol{E}_{test} - \boldsymbol{E}_{training}$ as far as possible [**?**].

Since ImageNet contains around 15 million training examples it has enabled the training of very large networks such as the 2014 winner GoogLeNet which had 22 layers [**?**]. In our case we are limited by the size of our dataset which contains a total of 6228 usable examples, many of which are practically identical due to being adjacent frames. A common method used to apply deep CNNs to tasks which lack large amounts of training data is to use a network pre-trained on a large dataset, typically ImageNet. Numerous studies [**?**, **?**, **?**, **?**, **?**] have shown ImageNet trained networks producing better results than randomly initialised networks in a variety of tasks, and also on depth data [**?**, **?**]. Typically the filters in the early layers are generic edge, corner or blob detectors which can be expected to generalise well to different tasks. In higher layers, as the spatial size of the input to each filter increases (due to pooling), more task specific representations are learnt.

2.3.1 Human Pose Estimation Using CNNs

Jain et al. [**?**] were the first to apply CNNs to human pose estimation. They trained multiple small three-layer networks to act as joint detectors with a separate network for each joint. Each network takes a small window of the full image as input. They are applied repeatedly with some overlap to produce a probability map of the joint's estimated location. They then use a spatial model which enforces consistent and valid global pose estimates from the estimated joint locations. In [**?**] they extend their system to videos using RGB plus motion features e.g. optical flow.

In a similar work Toshev and Szegedy [**?**] used a multi resolution approach to find 2D joint locations. They used the architecture of [**?**] first applied to the full image at a low resolution. They then apply another network to a higher resolution image of just the area around the locations of the joints as determined by the first network. This successively refines the predictions.

Li and Chan also measure 2D joint locations with a CNN. They use a single network with three convolution layers which they train for a pose regression and a joint detection

simultaneously, sharing all convolution filters between tasks. They show that training for the extraneous task of joint detection consistently improves the accuracy of the regression task. In [**?**] they extend their work to 3D joint position measurements. They also compare the original multi task training with pre-training on joint detection before fine-tuning on regression. They find little difference between the two but still show that the joint detection task consistently improves the regression performance. They show that the network produces reasonable position estimates even for completely occluded joints.

Pfister et al. [**?**] used the architecture of [**?**] (a 2013 ILSVRC winner) to regress 2D upper body joint locations from RGB video. They experiment with using an ImageNet pre-trained model but find results are improved when training from scratch, a result echoed in this work. They found their accuracy improved by 5% after performing background subtraction on their data, a method we also adopt. They also experiment with using multiple frames as input but find this gives only a modest 0.3% improvement.

Recently Belagiannis et al. [**?**] showed that convergence rates and final accuracy in HPE tasks can be improved by replacing the common L2/Euclidean loss function with a function that reduces the effect of outliers in the objective space. They state that the L2 loss functions give a disproportionally high weight to outliers which negatively effects the training procedure, reducing the generalisation ability and increasing the convergence times. They propose a new loss function, Tukey's biweight loss function, which acts to minimise the effect of outliers and show that employing this function improves results in better accuracy and convergence rates on 2D HPE task. This is an interesting finding which is especially relevant to this work where noisy skeletons produce a significant number of outliers. Unfortunately this paper was not discovered until after most of the work was completed. A direction of future work will be to implement this loss function.

3 Methods

3.1 Data Preprocessing

The dataset used in this project (SPHERE-staircase2014 dataset [**?**]) includes 48 sequences of 12 individuals walking up stairs, captured by a Asus Xtion depth sensor placed at the top of

the stairs in a frontal and downward-looking position.

It contains three types of abnormal gaits typical of lower-extremity musculoskeletal conditions. These include: freezing of gait referred to as stop $\times n$ where n is the number of freezes. Using a leading leg, left or right, in going up the stairs, referred to as LL and RL. All frames have been manually labelled as normal or abnormal by a qualified physiotherapist. There are 17 sequences of normal walking from 6 individuals and 31 sequences from the remaining 6 subjects with both normal and abnormal walking. The dataset contains a reasonable variation in body shape and appearance as can be seen in figure **??**.

Each example consists of the RGB image, the depth image and 3D joint position / skeleton data. Preprocessing is applied to each of these separately.

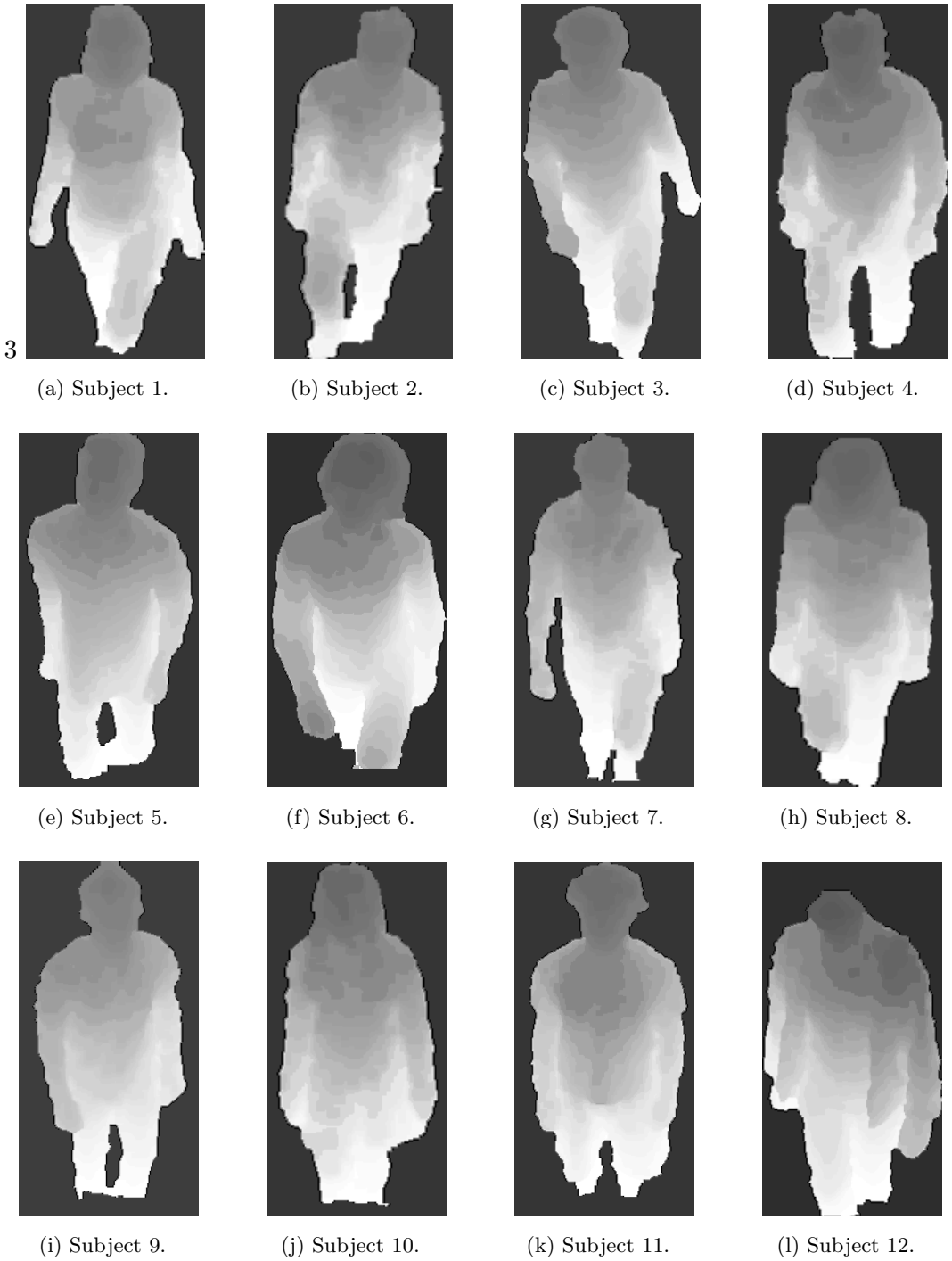
3.1.1 Skeleton Preprocessing

These processes were developed by the authors of [**?**] and are a pre-requisite for the manifold learning stage of that work and this. First the Skeleton data is smoothed over time to reduce the large amount of noise. Each skeleton is scaled to the same height, rotated to face forwards and translated to the origin. Each skeleton is compared with a neutral reference pose; a dissimilarity measure is computed using the sum of squared distances between corresponding joints, standardised by a measure of the scale of the reference shape. Any frames where this dissimilarity is greater than 0.1 are discarded. These are generally frames from the beginning and end of sequences, where the subject is outside the sensors optimum range, and any particularly noisy periods in the middle.

Specifically for this project we also take the mirror image of each skeleton. Combined with a mirror image of the depth image, this doubles our data.

Now the collection of remaining skeletons is given to the manifold learning stage. The method used is Diffusion Maps [**?**] with an adaptation to better handle the remaining noise and outliers present in the skeleton data. We refer to the original publications [**?**] and [**?**] for a full technical description. In essence Diffusion Maps retains the relative distances between data in the reduced space of the data. [**?**] analysed the quality assessment performance using 1,2,3,4, and 5 dimensional representations and concludes that a 3D representation is the most effective.

The set of data which is used in building the manifold determines its form. In the original



(a) Subject 1. (b) Subject 2. (c) Subject 3. (d) Subject 4.

(e) Subject 5. (f) Subject 6. (g) Subject 7. (h) Subject 8.

(i) Subject 9. (j) Subject 10. (k) Subject 11. (l) Subject 12.

Figure 8: The subjects contained in the SPHERE staircase 2014 dataset.

works [?] and [?] the authors used only the 'normal' sequences from the first 6 subjects. The manifold this produces is shown in figure **??**. We initially worked with a manifold built from all sequences abnormal and normal from each of the 12 subjects. This produces a slightly different manifold shown in figure **??**. Once we switched to using the final manifold we observed an increase in accuracy of around 20%. This shows that the form of the manifold can have a significant effect on the accuracy of the CNN, and therefore that these results for gait analysis may not be a reliable indication of performance on other movements and manifolds.

All results presented here after use the 2793 skeletons from the first 6 'normal' sequence subjects plus their horizontal flips to build the manifold, shown in figure **??**. The skeletons of the 31 sequences from the other 6 subjects (and their flips) are then projected onto the manifold (again, see [?] and [?] for details). Interestingly the flipped skeletons produce a manifold coordinate precisely equal to the non flipped version but with the first component negated. This suggests that each of the three components of the manifold are tied with the 3 regular spacial dimensions of the skeletons, although there is no guarantee of this with the Diffusion Maps method.

To illustrate the meaning of the manifold coordinates the skeletons placed at the 3 corners are shown in figure **??**. Generally the normal gait sequences follow paths across the dense quarter sphere shaped surface with the frame of maximal left and right knee flex occurring at the maximum and minimum of first coordinate. The second dimension, plotted horizontally in figure **??**, seems to measure the vertical distance between skeletons. The skeleton with raised knees occur near its minimal value and at its maximum we find elongated skeletons which are measured erroneously when subjects are very close to the camera but just survive the cut on skeleton dissimilarity. During processing of the images we remove these images from the dataset which results in the removal of most of the points which lie at this end of the manifold, as shown in figure **??**.

The meaning of the 3rd manifold coordinate is less clear. Figure **??** compares skeletons of minimum and maximum values for points of equal Y[0], Y[1], we are able to find no discernible pattern. This is reflected by the CNN as we find it far less accurate in this coordinate than the others. However this coordinate is somewhat subject specific, as seen in figure **??**, with each subject found to traverse the manifold with a Y[2] range smaller than that of the full dataset. We find that fine tuning our networks on the spare sequences of the subject being

tested improves results by 28%.

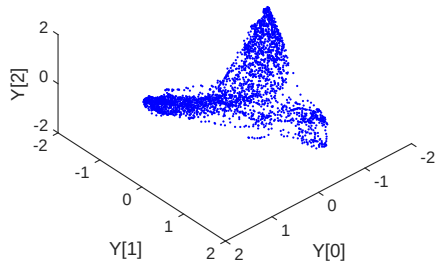
3.1.2 Depth Preprocessing

As described in section **??** depth images are generally incomplete and noisy. We first fill holes in the data using a simple method which iteratively paints in the maximum pixel values from the neighbouring cells. We initially experimented with using the method of [**?**] which combines noise filtering and hole filling. However the version that we accessed did not use the motion detection and segmentation components. In practise we found that the quality of the filled and filtered depth maps produced by this version were inferior to those produced by the simple method. [**?**]'s method also required far longer processing times, 1 week for the full set as opposed to a 20 minutes for the simple method.

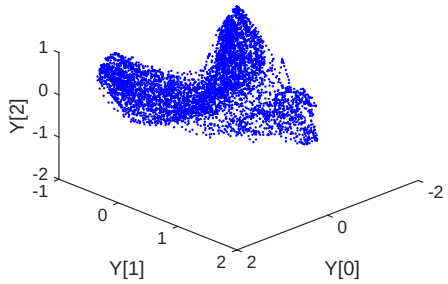
We then perform background subtraction on the filled depth images. We employed the C++ BGS library [**?**] and tested each of the provided methods on our data. For all methods some initial set of background frames at the beginning of the sequence were required to achieve satisfactory results. The best results were achieved using the dp/DPZivkovicAGMMBGS method which implements [**?**].

We found that the foreground masks produced often mistook sensor noise for moving objects. To fix this problem we apply a simple cleaning procedure to the masks which takes only the largest collection of positive pixels. This discards the small noise related blobs. We also apply a small erosion and dilation to get rid of noise effects connected to the largest blob. There are also problems correctly identifying the feet of the subjects since the floor is at approximately equal depth and therefore the same colour in the depth image. The masks generally only extend to around the mid point of the shin. For some subjects feet can appear mid stride when raised high enough from the ground to be distinguishable. We also attempted using the RGB images to improve extraction of the feet. However most subjects are wearing dark coloured shoes which did not differentiate well from the colour of the stairs, meaning the results were just as poor as for the depth, with additional issues in other parts of the images where clothing matched the colour of the walls. Additionally the subtraction fails badly on some frames, particularly towards the end of a sequence. These images are discarded.

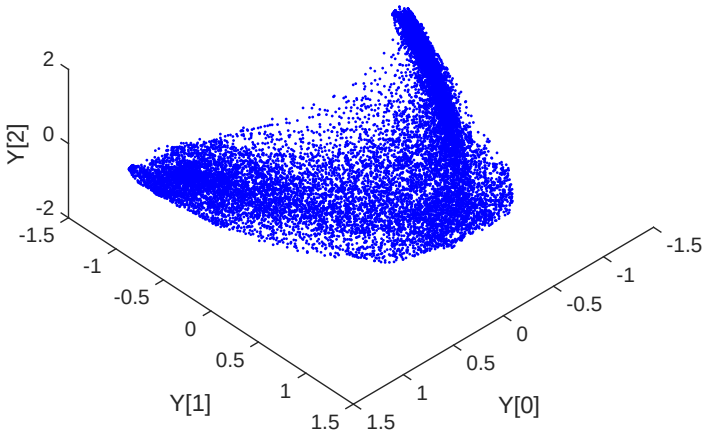
We then take the cleaned mask and extract the foreground containing the human figure from the depth image. In some cases patches of background get included around the edges of



(a) The manifold produced by normal sequences of subjects 1-6. This was the form used in [**?**]. We connect points from neighbouring frames for better visualisation of the structure.



(b) The manifold produced by normal sequences of subjects 1-6 including their horizontal flips. This was the form used in this work.



(c) The manifold produced by all sequences including their flips. The larger variation in the z component (the vertical axis in the figure) led to lower network accuracy and incorrect normality analysis.

Figure 9: Shows the pose manifolds used in this work.

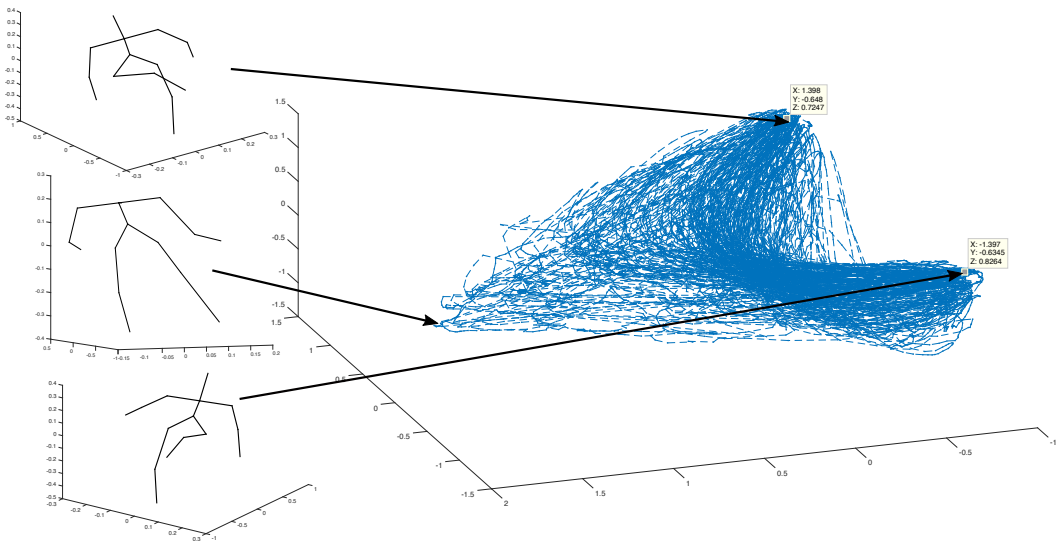


Figure 10: Shows the skeleton to manifold mapping. Normal gait sequences trace paths between the two corners on the right, with the postion of maximal knee flex the turning point. The left most coner of the manifold consists of elongated skeletons which tend to be measured when the subject is too close to the sensor.

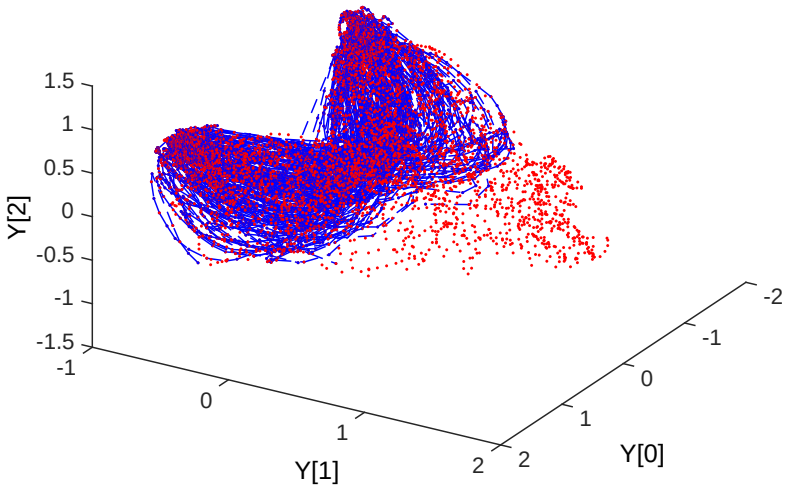
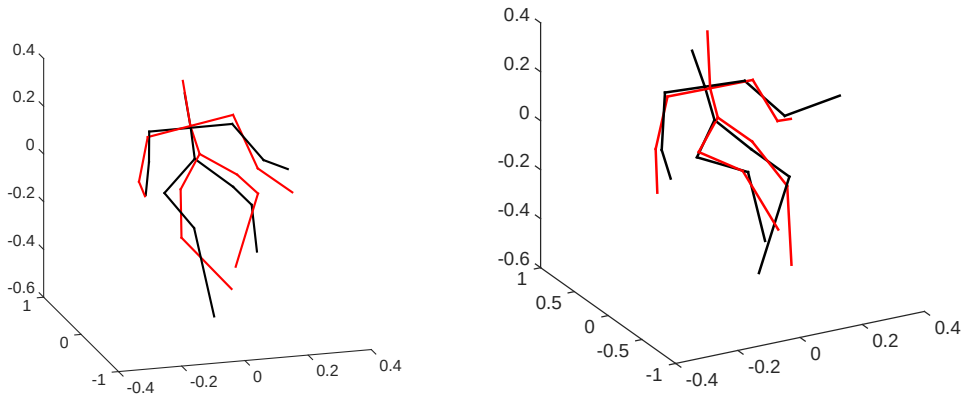


Figure 11: Shows all manifold points in red. After image pre-processing we retain only those in the blue area.



(a) Difference of 0.4099

(b) Difference of 0.503.

Figure 12: Shows skeletons within 0.02 of the same Y[0],Y[1] postition but with maximum difference in Y[2]. Red skeletons are mimimum Y[2], black are maximum. We find no clear difference between the Skeleton of minimum and maximum Y[2]. This is reflected by the CNN which measures this coordinate with a lower accuracy than the first two.

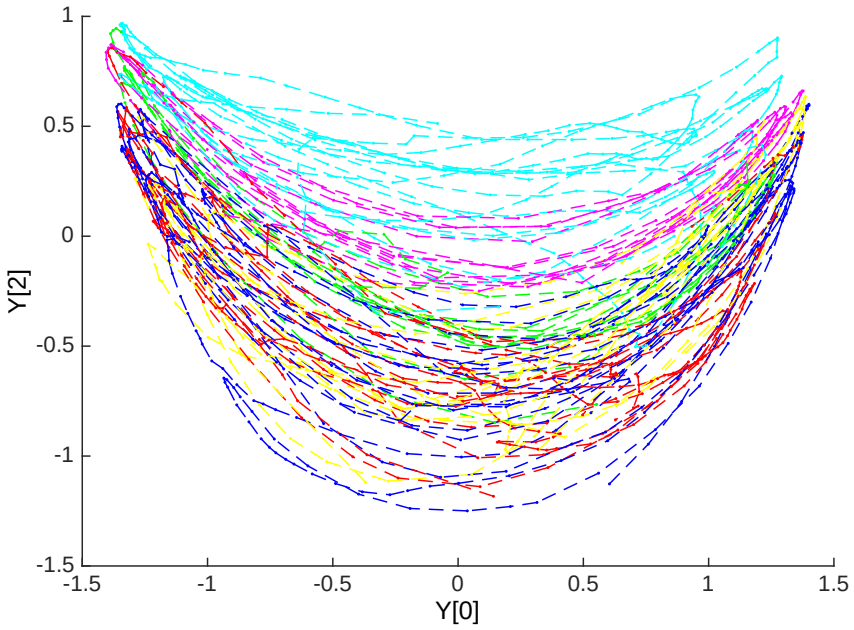


Figure 13: Each normal sequence of the first 6 subjects plotted in a different colour. We observe that Y[2] coordinates for each subject tend to remain within a sub-region of full range. This finding promoted the use of subject finetuning which improves the accuracy of Y[2] predictions.

(a) Typical image before hole filling.

(b) Filtered and filled using a simplifed version of [**?**].

(c) Filled using max fill method.

Figure 14: A comparison of hole filling methods. We use the max fill method.

the mask. We remove any pixels which have values greater than two standard deviations from the mean of the foreground. Due to the perspective of the camera the depth values of the of the foreground increase towards the feet of the subject, therefore if this cut is too strict we begin to loose valid pixels near the feet. To avoid this whilst still removing all background pixels from the rest of the image we select only the top 75% of the figure and apply a more stringent cut.

Since all skeleton data was scaled and transformed to a common point we do the same to the depth data. We normalise the mean depth values of each image by finding the mean depth value of a small region near the subjects waist. Using only this region gives an accurate indication of the distance of the subject from the camera rather than transient effects such as arm swing. This value is subtracted from all foreground pixels before all being scaled to 0.5.

Next we crop the background area from each image so that each is left with the average width to height ratio of the full dataset which is 0.504.

We then try to normalise the position of the figure within the image. There are two issues with the data which make this a necessary step, firstly the length of lower leg included in the masks varies from frame to frame, secondly there are a number of sequences in which the head of the subject leaves the camera's field of view during the middle of the ascent. Without fixing this problem the scale of the subjects in the images changes depending on how much of them was captured in the mask, which would mean the network would be required to learn an invariance to scale. In general we developed these preprocessing procedures to reduce the amount of invariance the network is required to learn. To fix this we try and identify the position of the shoulders. The width of the mask as a function of height in the image was found across the full dataset, the point of greatest curvature was determined to lie at a mask width of 0.65 of the full image width (after initial cropping). The point at which the the width of the mask exceeds this value is defined as the shoulders. From this we find the position of the shoulders in each image. The mean position of the shoulders in all non-headless images was found to be 0.22 of the full image height. Then each image was rescaled, adding additional pixels of background so that the position of the shoulders occurs at exactly 0.22. If the initial shoulder position was less than 0.22 then we assume the image is missing part of its head and rows are added to the top, if it is greater than 0.22 we assume it misses more leg than usual and rows are added to the bottom. Whilst this doesn't recreate the actual foreground

pixels of the legs or head, it does mean that the foreground pixels for each intact body part do occupy roughly the same position in the images. One limitation of this method is that it is difficult to handle images which lack both head and legs. We attempt to deal with this by storing the mean number of rows added to the bottom of the image across the sequence. Then if we find on the next frame that we are missing rows from the top (i.e. part of the head is missing) then we continue to add the mean number of rows to the bottom.

Next we adjust the colour balance of the image. Initially we used the basic scaled depth values and a black background. Analysis of the activations produced by the first layer filters of an ImageNet pre-trained network on this type of input, shown in figure **??**, seemed to indicate that depth information was not being extracted effectively. ImageNet first layer filters, shown in figure **??**, generally respond to edges. It is clear that the very small differences in depth between parts of the body will produce far smaller activations than the huge difference between the figure and the black background. To encourage the network to focus on the smaller edges we tried setting the background of the images to the mean depth value around the waist of the subject, as shown in figure **??**. However after training until convergence the network seemed to have discarded most of the filters activations rather than adjusting to them as is seen in **??**. Finally we used histogram equalisation to increase the differences in depth values within the figure. We use a background value of 0.82 of the mean value at the subjects waist as this was slightly less than the minimum value of any valid foreground pixels seen in the dataset. This results in the largest spread possible in the foreground depth values without ever having a foreground value at a colour darker than the background. This scheme seems to achieve the desired result of allowing interior depth values and edges to be preserved through the first convolution layer as seen in figure **??**.

Finally, we remove images from the beginning and end of sequences where the subject is rotating or is very close to the camera as these are generally inaccurate and also because there are so few of these frames that they represent rare outliers in the dataset.

A common pre-processing step when working with CNNs is to subtract the mean image of the training data (i.e. the mean value of each pixel) from each image. This is supposed to speed up training for reasons detailed in [**?**, **?**, **?**]. After testing the accuracy of the networks trained on such mean subtracted data we found that this operation actually decreased the performance as shown in figure **??**. However this test was only conducted on an ImageNet

(a) Initial scheme. (b) Mean background method. (c) Final scheme. (d) Pre-processed RGB

Figure 15: Shows the input image colour schemes tested with the CNN. We found (c) produced the largest responses in the first layer filters of a pre-trained network, shown in figure **??**

pre-trained AlexNet (see section**??**), we have not studied the effect when training from scratch.

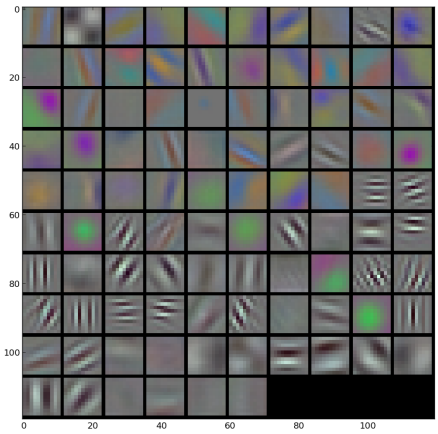
3.1.3 RGB Pre-processing

For RGB images we applied the foreground masks retrieved from the corresponding depth images and then applied clipping and scaling in the same manner as the depth images. The result is shown in **??**.

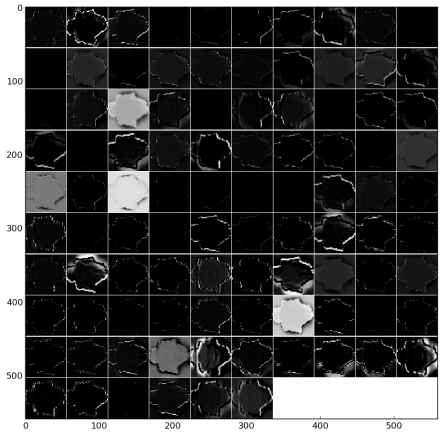
We measured the accuracy of an ImageNet pre-trained AlexNet (see section**??**), which requires a 3 channelled input image, when using the RGB images only, the depth image replicated for each of the 3 channels only, and a combination of the two which consisted of 1 channel of the average of red and green, 1 channel of green and blue averaged and the depth in the final channel. As is shown in figure **??** the replicated depth was shown to produce the most accurate regression. All other results reported here use a pure depth input image.

3.2 Software

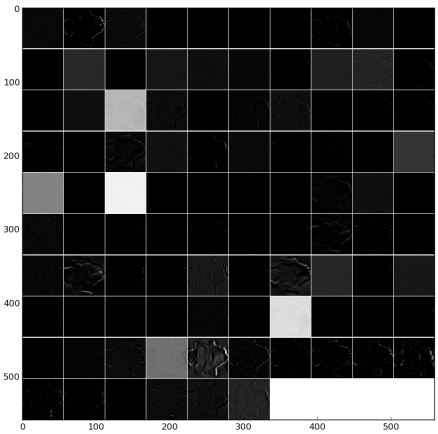
We use the open source CNN library Caffe [**?**] which is common among researchers including [**?**, **?**, **?**, **?**, **?**, **?**, **?**].



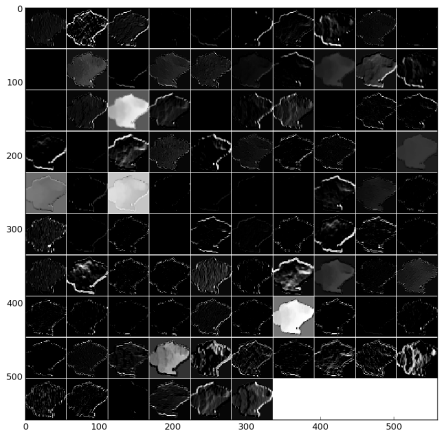
(a) The filters in conv1 of a imagenet pre-trained then finetuned AlexNet [**?**].



(b) Initial colouring scheme



(c) Mean background method



(d) Final scheme

Figure 16: The activations produced by the first layer filters (a) for each of the colour schemes of figure **??**.

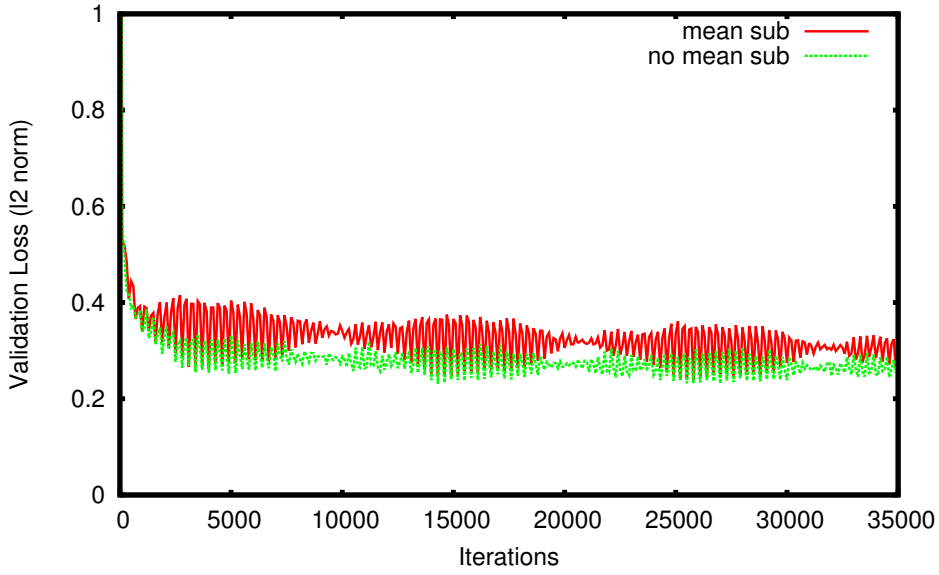


Figure 17: Shows the network accuracy against number of training examples when using mean subtracted data and non mean subtracted data. The accuracy of network predictions was seen to decrease when using mean subtracted data, hence we abandon this common practise.

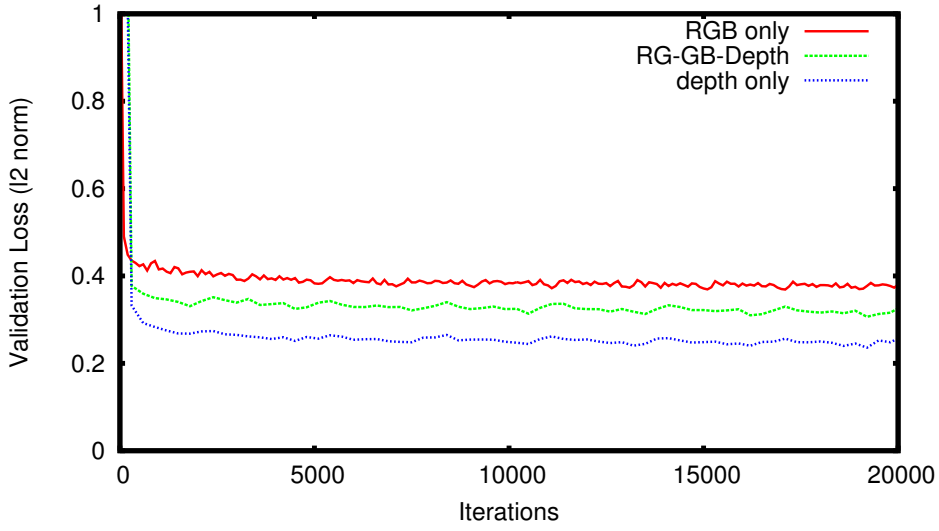


Figure 18: Shows the network accuracy when using the RGB images only, the depth image replicated for each of the 3 channels only, and a combination of the two which consisted of 1 chanel of the average of red and green, 1 channel of green and blue averaged and the depth in the final channel. We use depth only for final results.

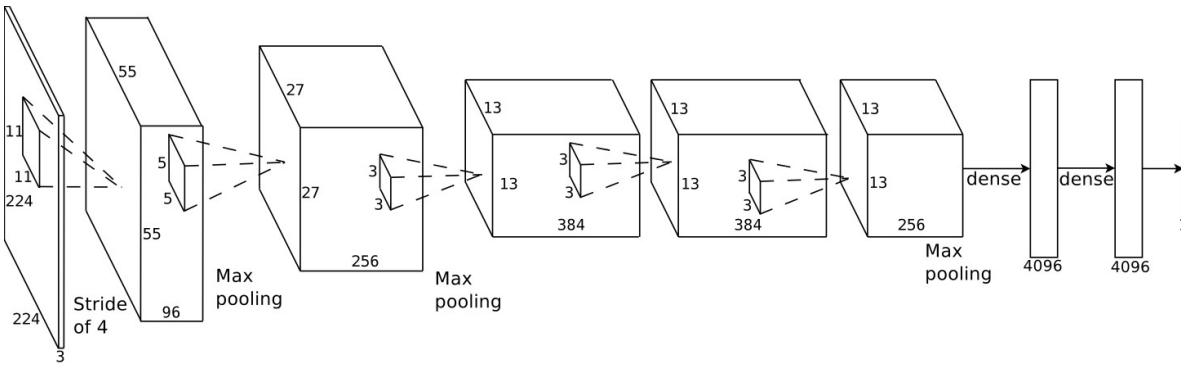


Figure 19: The architecture of [**?**] is used.

Caffe requires datasets to be in certain formats for training. When using multi-dimensional labels, as in our case, you are constrained to using a HDF5 formatted [**?**] dataset.

One disadvantage of this is that automated shuffling options are not available as in the other dataset formats. Shuffling training examples is important for achieving the best training results in the shortest times. When training, each adjustment to the network's parameters is proportional to the error on the last example. If the network is shown each ascent sequence in order the errors between each consecutive frame will be small since they contain very similar images and poses. Therefore we will waste lots of iterations making small updates. Since the HDF5 input layer in Caffe doesn't provide an automated shuffling we have to pre-shuffle the images and labels before storing them.

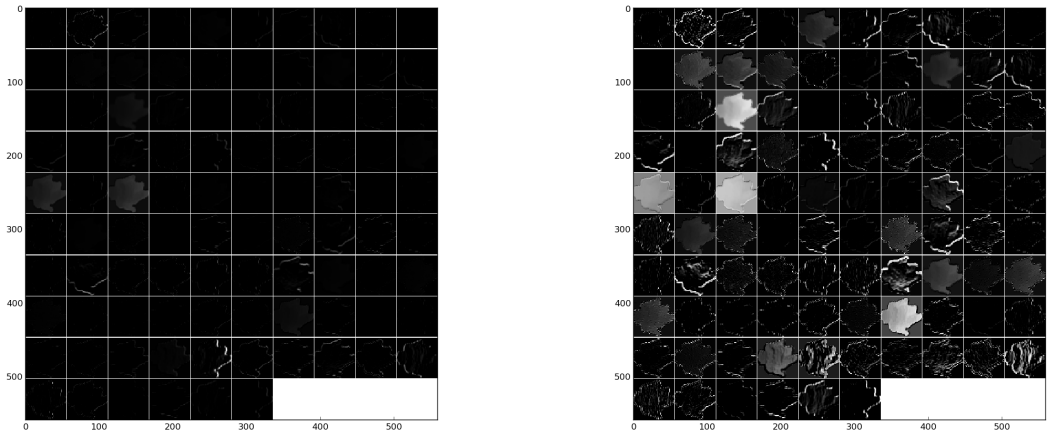
3.3 Architecture

The architecture used, shown in figure **??**, is a version of that which won the 2012 ILSVRC [**?**]. It consists of 5 convolutional layers, the 1st, 2nd and 5th of which are followed by max pooling layers.

This network also features Local Response Normalisation (LRN) layers after the 1st and 2nd convolution layers. These normalise the values of each filter's activation with respect to the others in the same spatial position. Each activation is divided by

$$(1 + \frac{\alpha}{n} \sum_{i}^{n} x_i^2)^{\beta} \tag{4}$$

where x_i are the activations of filter i at the same position. n, α and β are adjustable parameters,



(a) The activations before LRN.

(b) And after

Figure 20: Shows the effect of Local Response Normalisation on 1st layer activations.

which we leave unchanged from those used by [**?**] at 5, 0.0001 and 0.75 respectively. The effect of this operation on our data is shown in figure **??**.

To adapt this network for regression we replace the final fully connected layer with a 3 element layer which produces our final predicted pose vector. The Softmax loss function is replaced with an Euclidean loss which computes the l2-norm between this final vector and the label.

One advantage of using this architecture is that we can begin training from the weights trained on ImageNet before fine tuning on our smaller dataset. We tested the networks performance under 3 of these initialisation schemes with all other parameters fixed: 1) using the pre-trained weights for all layers, 2) using the pre-trained weights only on the first two convolution layers with a random initialisation on the others and 3) using a full random initialisation. We found that using imageNet weights in all layers significantly hindered the final performance as shown in **??**. Schemes 2 and 3 produced very similar results, with scheme 2 having slightly better performance at some points earlier in the training. We use scheme 2 for our final results.

We also experimented with other architectures including VGG-S [**?**] the 2014 ILSVRC 2nd place model which has been shown to generalise better to other tasks than any other

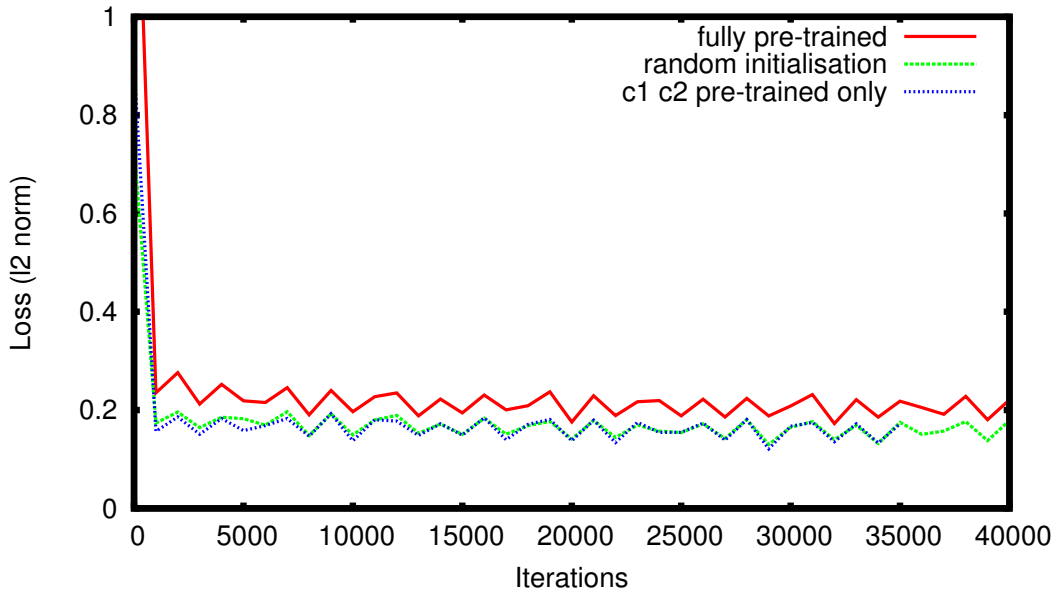


Figure 21: Shows a validation set loss for various pre-training schemes. We find that begining training from ImageNet pretrained weights throughout the network hinders performance. Using pre-trained weights only in the first two layers produced the best results. We use this strategy for our final tests.

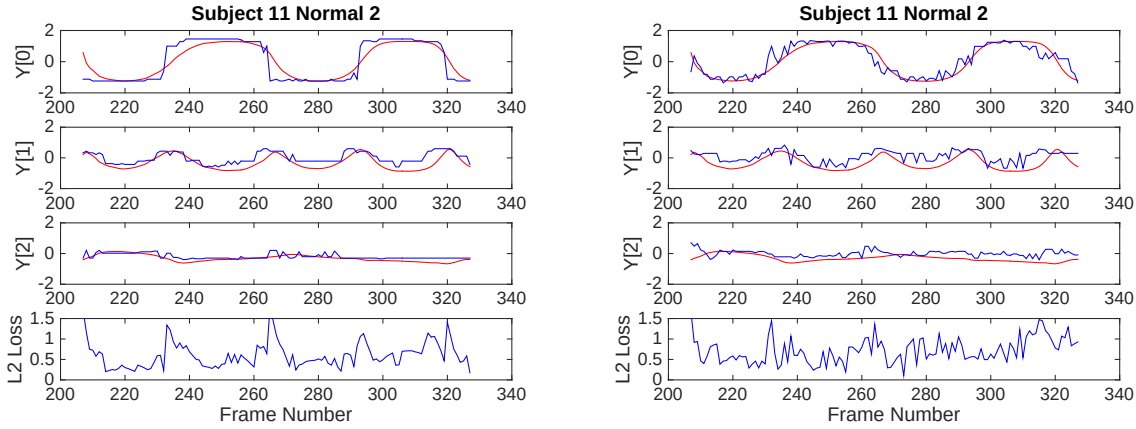
architecture [**?**]. However this deep 19 layer network was far too memory intensive. It could only run on our 2GB GPU with a batch size of 1 which produced extremely noisy losses and would not converge.

We also experimented with various custom architectures including: all convolutional architectures which forego pooling layers, using convolutional layers with 2 pixel strides instead, with the idea that this would increase the networks spatial precision since max pooling layers discard precise spatial information. Simple 2 and 3 layer architectures, based on the idea that our data is all very similar in appearance and therefore does not require very high level features. Architectures with filters sized to match the features we hoped them to extract e.g. initial layer filters were slightly wider than the edge effects in the image to avoid the noisy activations we see in figure [**?**], 2nd layer filters were sized to match arm and leg widths and 3rd layer filters to match shoulders and head sizes etc. However, we found none of these architectures to improve accuracy over that of [**?**].

3.3.1 Regression Vs. Classification

Regression tasks are generally harder to optimise CNNs for than classification tasks [**?**]. With a regression task we require one specific output from the network for each example. With a classification task, using a Softmax Loss, the exact output value is not important, only the relative values. Another problem with regression is the effect of outliers on training as we discussed with reference to [**?**] in section **??**. An additional benefit of classification is that you get a distribution over the outputs which gives an indication of confidence not possible with pure regression.

To this end we experimented with posing our problem as a classification task by binning each component of the pose vectors into a class. We experimented with a range of numbers of classes. Figure **??** plots the predicted pose vectors after being converted back to the mid value of the predicted class, as well as the L2 norm between this value and the labels (note this was not the loss used in training the network, but is calculated afterwards for a measure of accuracy). We find that 51 classes produces a mean loss of 0.60 and 500 classes a mean loss of 0.71. Our final regression network outperforms both of these with a loss of 0.13, this result is displayed in figure **??**. These results suggest that this approach is far less suitable than pure regression.



(a) With 51 classes per component. Mean loss = 0.6035 std = 0.3154.

(b) With 500 classes per component. Mean loss = 0.7141 std = 0.3054.

Figure 22: The top 3 graphs plot the 3 components of the pose vector for the labels (red), and the the network prediction (blue) for a classification network. The 4th plot shows the error measured as the distance between the labels and prediction. A mean error of 0.13 for this sequence was achieved using pure regression.

Inspired by the results of Li and Chan [**?**] who found that regression accuracy was improved by joint training with a joint detection task; we also attempted joint training of classification and regression tasks in various forms. These included: performing the classification off of early layers in an attempt to force gradient terms to the lower layers. A sub-net which classified the first component and shared features with the regression network. Both shared and separate fully connected sections for regression and classification. However, across all these studies we found no improvement in accuracy over pure regression. We also found that the more complicated architectures were prone to diverging during training.

3.4 Training Details

Following skeleton selection and depth pre-processing we are left with 6228 distinct examples, plus their horizontal flips. So as to best utilise our data we cross validate training 6 networks with pairs of adjacent subjects withheld for testing in each i.e. 1 and 2, 3 and 4, 5 and 6 etc.

Each of these networks was trained for 50000 iterations with a batch size of 25 examples on a GeForce GTX 750 GPU taking almost 7 hours each. We observed no overfitting effects

even after 700000 iterations. However results did not improve beyond 50000 iterations. The batch size was dictated by the 2GB memory limit of the GPU however it seemed reasonably suitable with only a small amount of noise in the loss curves as seen in figure **??**.

Additionally we present subject fine tuned results where each of these trained networks is further trained for 10 epochs (10 times through the training data) on the remaining non-test sequences of that same subject. We found that overfitting did occur for these small training sets. The optimum results required tailoring to the size of the training set rather than a fixed number of iterations.

In both cases we use the adaptive learning rate method AdaGrad [**?**] which scales the learning rate for each parameter based off the size of its previous updates. We found in practise that this method achieved a roughly equal loss to that of stochastic gradient descent with momentum and to Nesterov's accelerated gradient method. AdaGrad has the benefit of removing the need to tune for hyper parameters such as the learning rate and momentum. We also apply a weight decay of 0.005. We found that altering this had a negligible effect on the accuracy of the network so chose to keep it at this commonly used value.

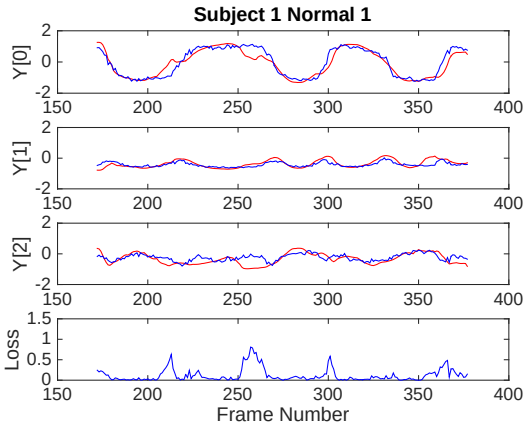
For each subject we present results for their longest sequence since longer sequences make the SPHERE software's gait cycle time analysis more accurate. In the case of subjects 9, 11, and 12 we test one normal and one abnormal sequence taking the longest one of each.

4 Results

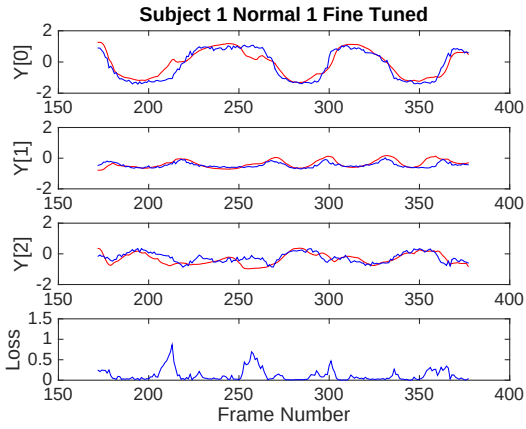
We measure the accuracy/error by the same loss function used to train the network i.e. the distance (L2 norm) between the predicted pose vectors, always shown in blue, and the ground truth labels shown in red. Across all tested sequences we find an average error of 0.1565 for the non subject fine tuned models which is reduced to 0.1129 after fine tuning.

For the first 6 subjects we present only accuracy data, shown in figure **??**, since these sequences are used by the SPHERE software to build the normality model.

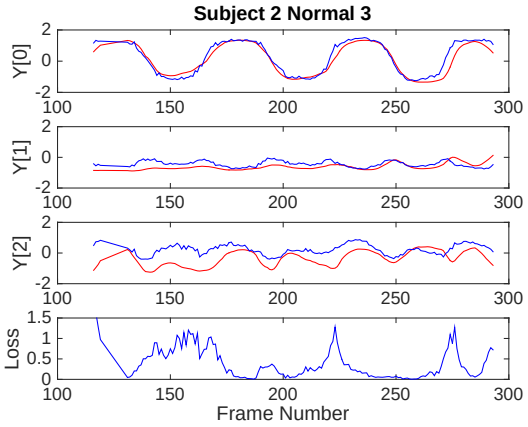
For subjects 7-12 we present both accuracy data and pose and dynamics quality scores, shown in figure **??**. These quality scores were computed for us by the authors of [**?**] using the same methods and models (trained from the skeleton data) of that work.



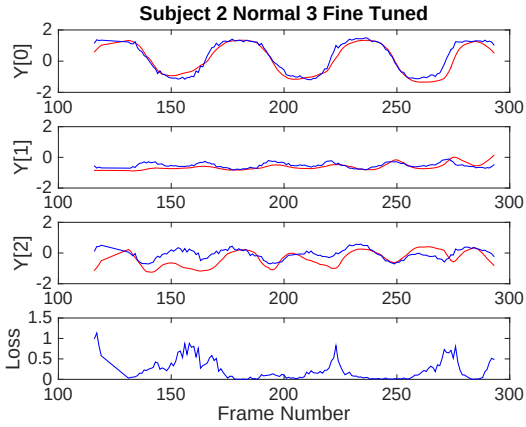
(a) mean loss = 0.1207, std = 0.1604.



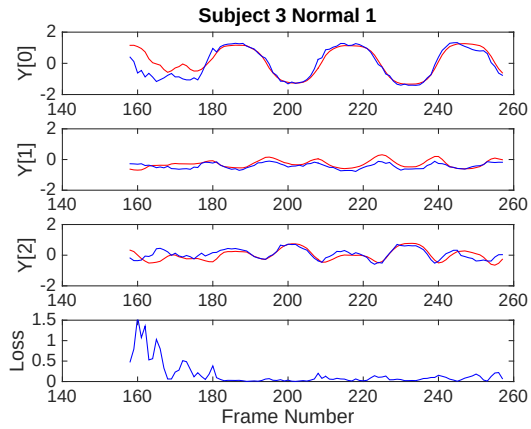
(b) mean loss = 0.1227 std = 0.1497.



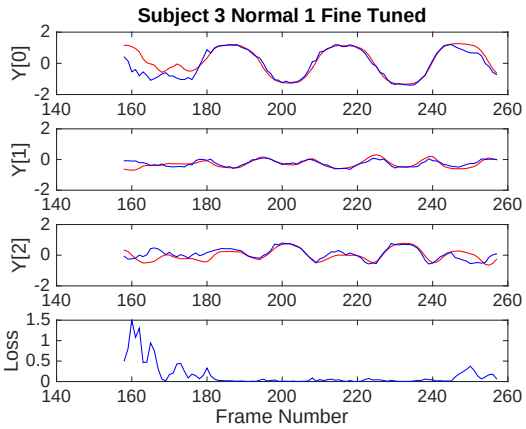
(c) mean loss = 0.3859, std = 0.3678.



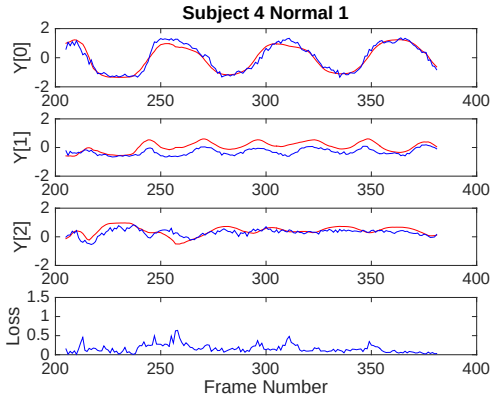
(d) mean loss = 0.2280 std = 0.2457.



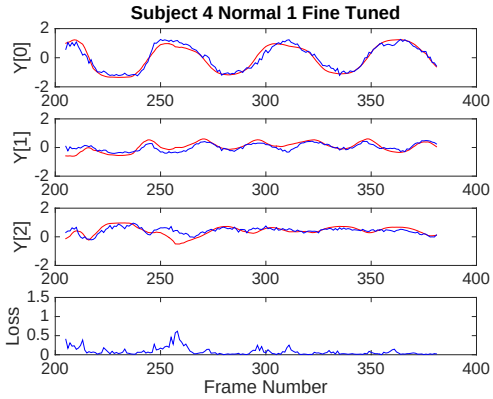
(e) mean loss = 0.1648, std = 0.2772.



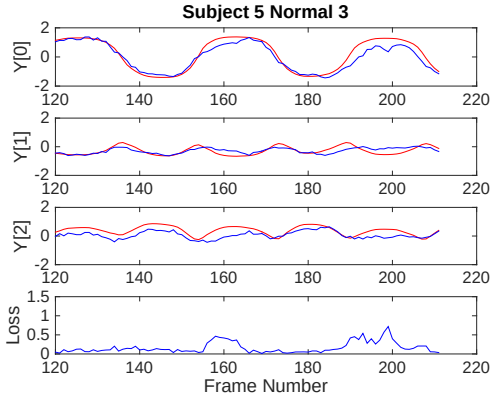
(f) mean loss = 0.1465 std = 0.2713.



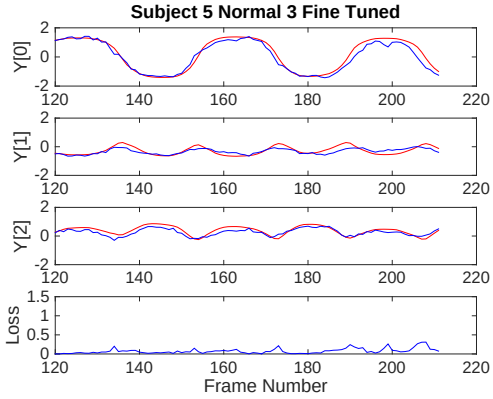
(g) mean loss = 0.1613, std = 0.1109.



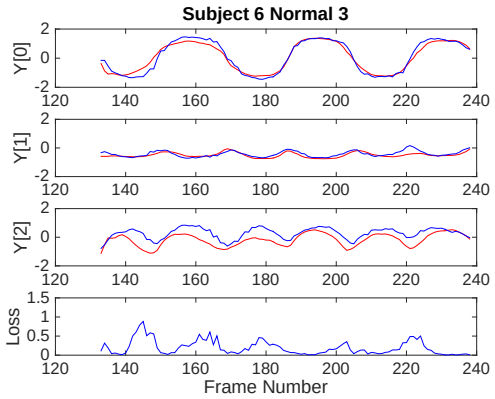
(h) mean loss = 0.0852 std = 0.1030.



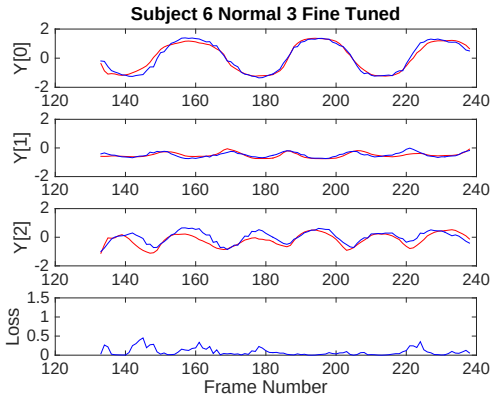
(i) mean loss = 0.1640, std = 0.1450.



(j) mean loss = 0.0751 std = 0.0751.

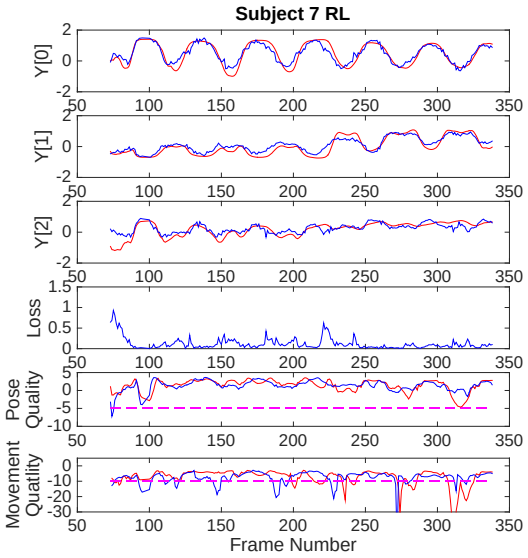


(k) mean loss = 0.1952, std = 0.1884.

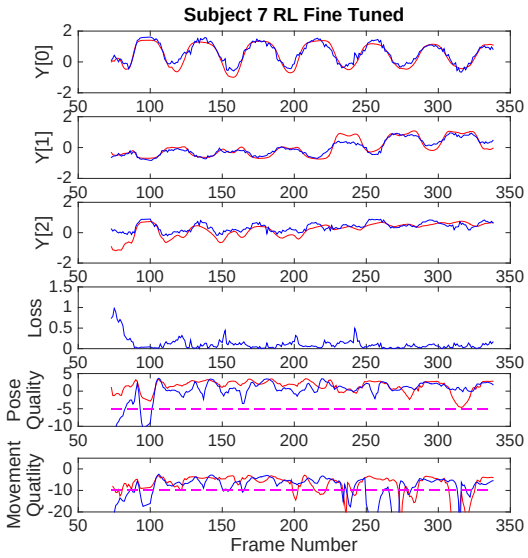


(l) mean loss = 0.0873 std = 0.0970.

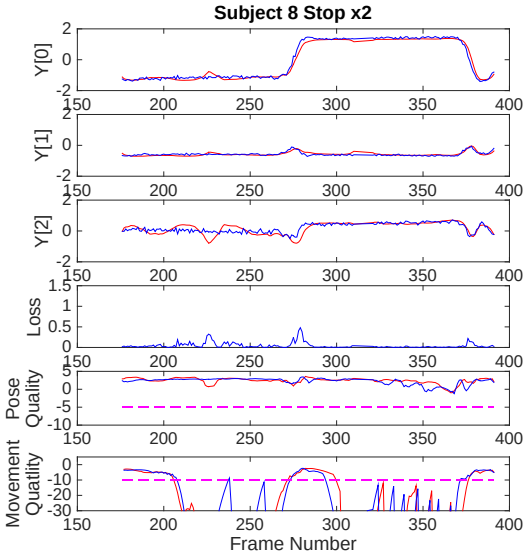
Figure 21: Top 3 graphs show the 3 components of the pose vector for the labels (red), and the the network prediction (blue). The 4th plot shows the error measured as the distance between the labels and predictions. Fine-tuned results are produced by networks which have been trained with spare sequences of the subject being tested.



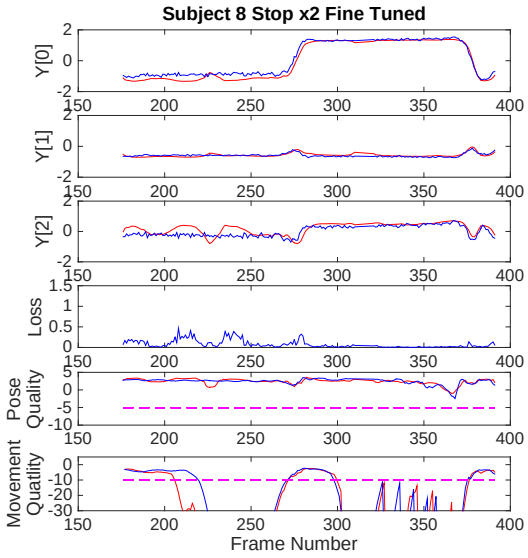
(a) mean loss = 0.1208, std = 0.1388.



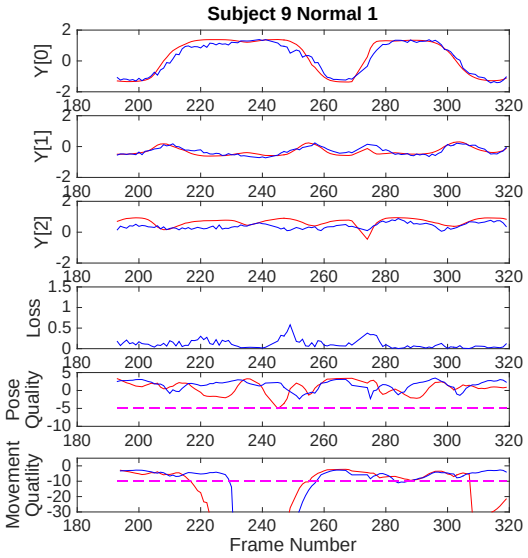
(b) mean loss = 0.1133 std = 0.1439.



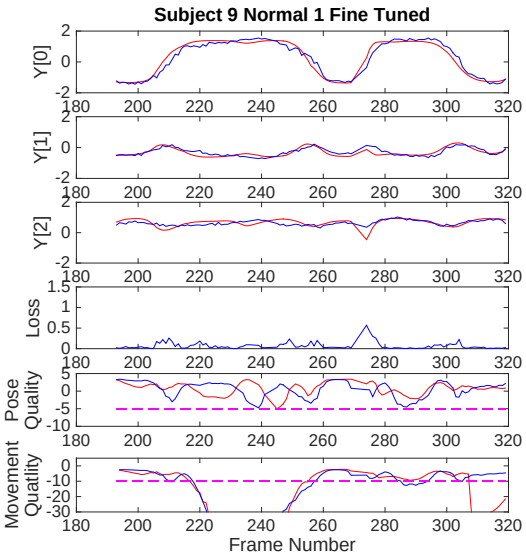
(c) mean loss = 0.0457, std = 0.0720.



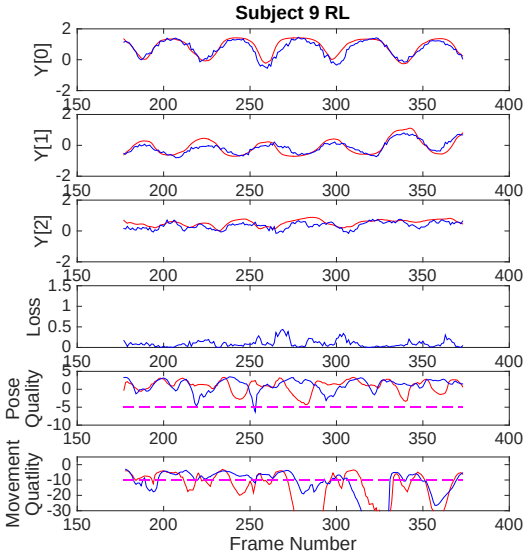
(d) mean loss = 0.0795 std = 0.0950.



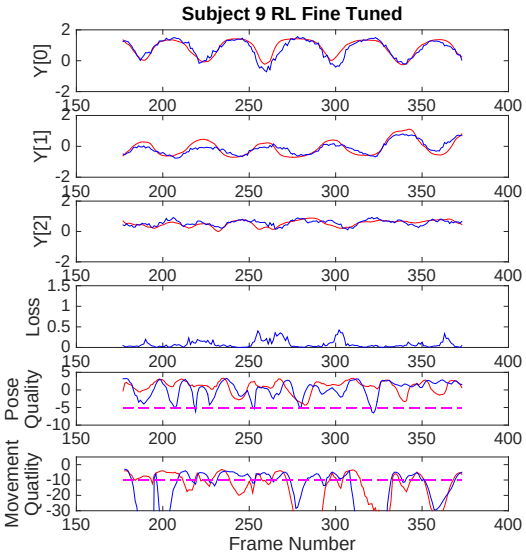
(e) mean loss = 0.1110, std = 0.0995.



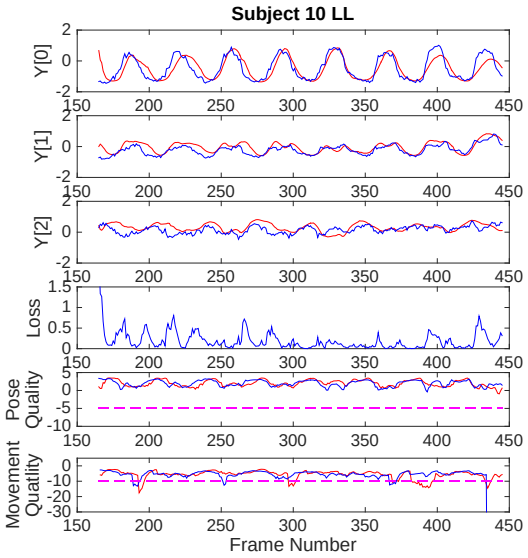
(f) mean loss = 0.0692 std = 0.0852.



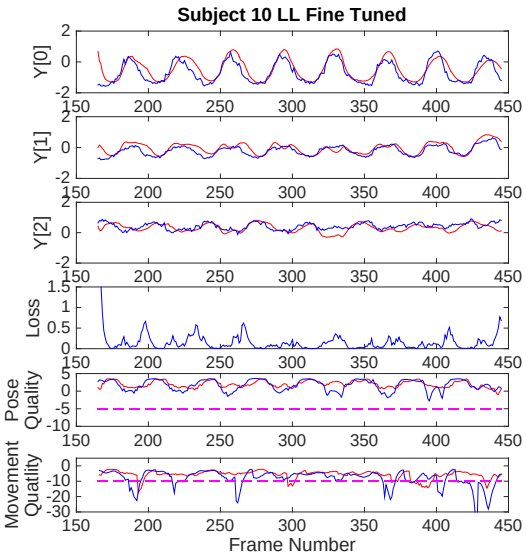
(g) mean loss = 0.0944, std = 0.0875.



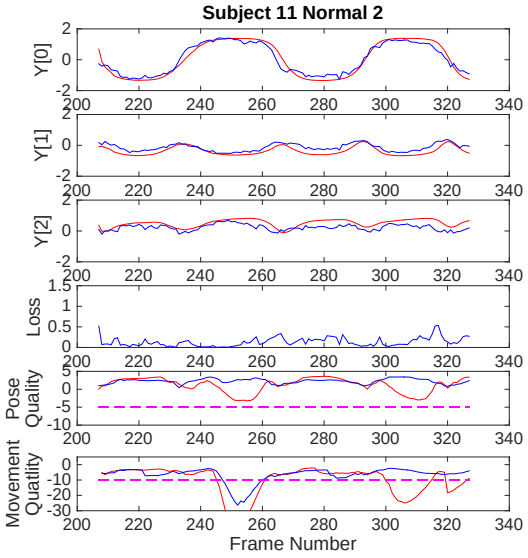
(h) mean loss =0.0821 std = 0.0917



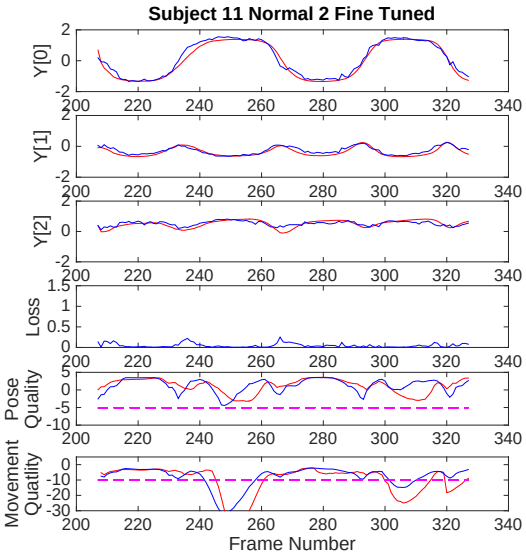
(i) mean loss = 0.1937 std = 0.2353.



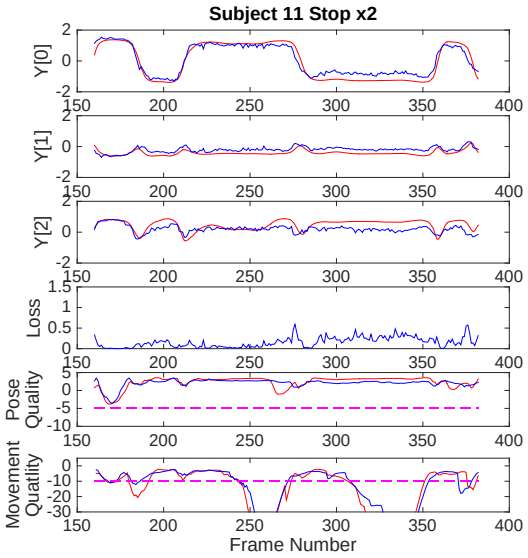
(j) mean loss =0.1575 std = 0.2448



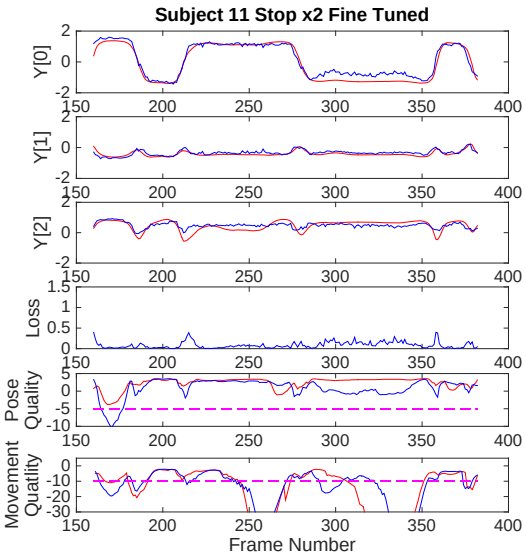
(k) mean loss = 0.1337, std = 0.1083.



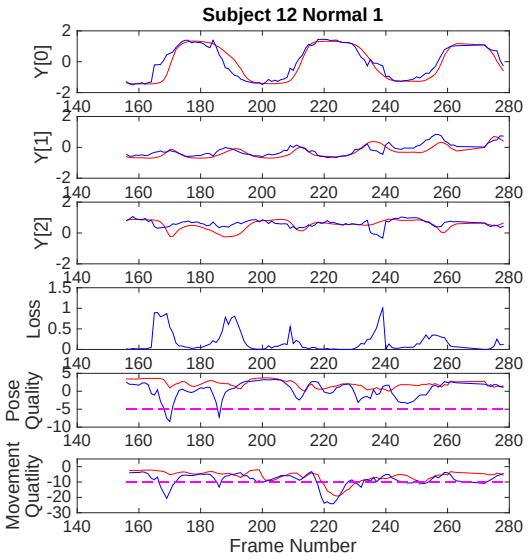
(l) mean loss = 0.0474 std = 0.0496



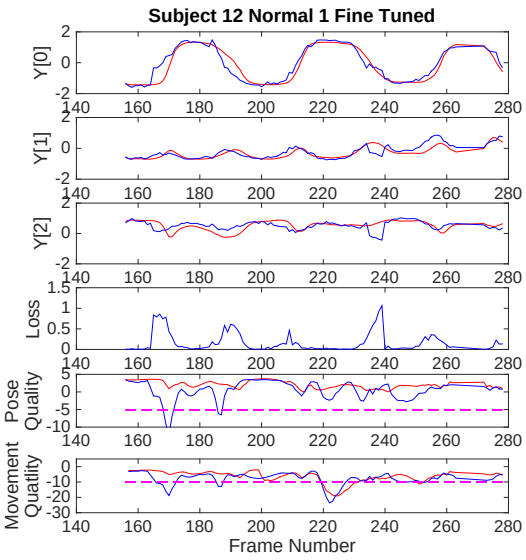
(m) mean loss = 0.1547, std = 0.1231.



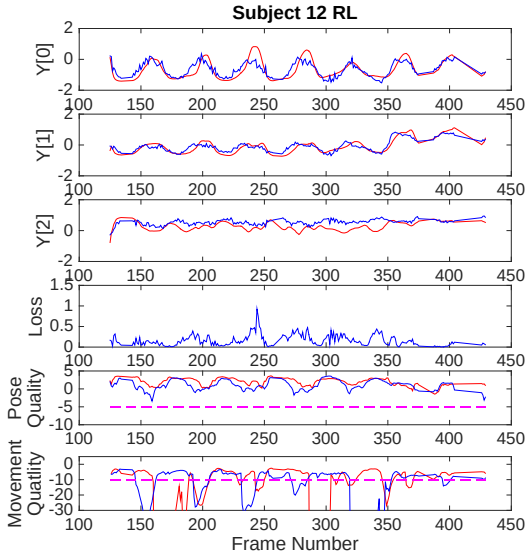
(n) mean loss =0.0867 std = 0.0792



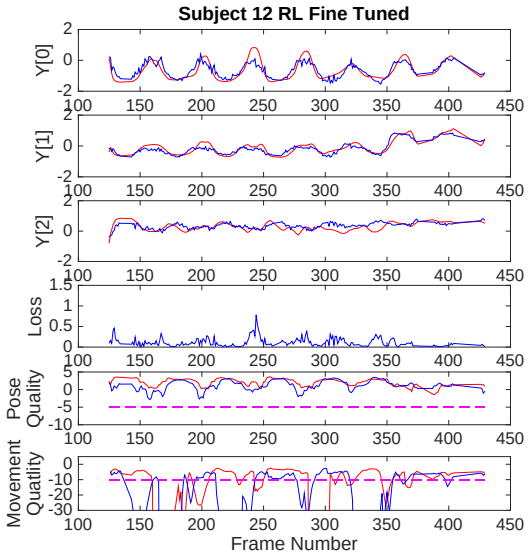
(o) mean loss = 0.1893, std = 0.2492.



(p) mean loss = 0.1673 std = 0.2368

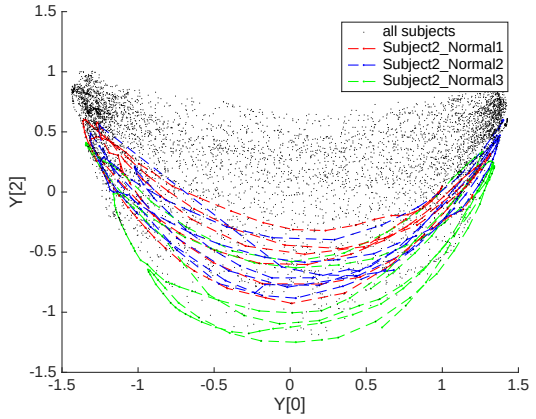


(q) mean loss = 0.1611, std = 0.1334.

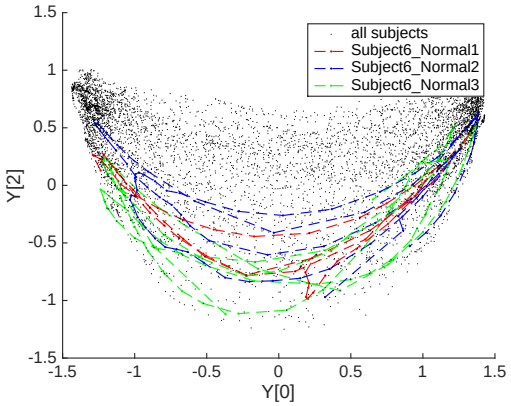


(r) mean loss = 0.1097 std = 0.1040

Figure 17: Top 3 graphs show the 3 components of the pose vector for the labels (red), and the the network prediction (blue). The 4th plot shows the error measured as the distance between the labels and predictions. Fine-tuned results are produced by networks which have been trained with spare sequences of the subject being tested. 5th and 6th plots show the movement quality measurements of [**?**] when given the network's prediction (blue) and the original results of the labels (red). The magenta dashed line shows the empirically determined abnormality thresholds.



(a) Subject 2. Fine tuning on Normal 1 and 2 produced a decrease in error of 0.1579 in 3.



(b) Subject 6. Fine tuning on Normal 1 and 2 produced a decrease in error of 0.1079 in 3.

Figure 18: Shows a plot of the skeleton/label data for each sequence of the subjects in which fine tuning produced the greatest decrease in the error of the network's predictions.

4.1 Fine-Tuning

For 13 of the 15 tested sequences fine tuning reduced the mean error. The greatest decrease was seen in that of subject 2 (figures **??** and **??**) with the error decreasing by 0.1579. In the non fine tuned case the predicted $Y[2]$ component appears to follow the form of the labels but with a slight offset producing a large and consistent error across the sequence. Looking at plots of the label data for these sequences in figure **??** we see that the tested sequence Normal3 contains the minimum $Y[2]$ points in the whole training set. This explains why the network was not able to measure this low $Y[2]$, since it has not been trained with any points with these values of $Y[2]$. The other subject 2 sequences used for fine tuning also have paths below the mean $Y[2]$ of the rest of the training data which explains why the fine tuning helps to reduce the predicted $Y[2]$. This same explanation also seems to apply to the second most effective fine tuning results, shown in **??**; that of subject 6 for which the error was reduced by 0.1079. These results suggest that with a more even spread of training data we could expect improved accuracy, with or without fine tuning.

For two sequences: Subject 1 Normal 1 and Subject 8 Stop $\times 2$ the error increased after fine tuning.

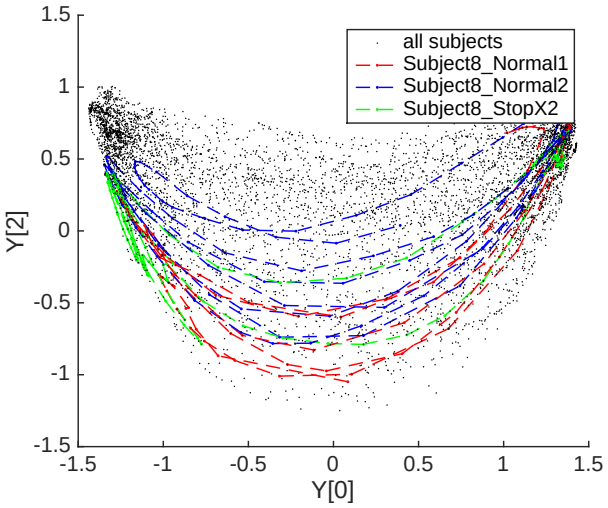


Figure 19: Shows a 3D plot of the skeleton data for Subject 8. Fine-tuning on Normal 1 and 2 produced an increase in error of 0.0338 in the Stop x2 sequence.

For Subject 8 (figures **??** and **??**) fine-tuning produced a decrease in the predicted $Y[2]$ and an increase in $Y[0]$ particularly for the first frozen pose (the tested sequence was a Stop $\times 2$). Both these changes took the predictions away from the ground truth, increasing the error. Looking at a plot of the training/label data, shown in figure **??** it seems the spread in $Y[2]$ for this subject was larger than for most other subjects. However the accuracy both before and after fine-tuning was well below the mean of the rest of the data.

In the case of Subject 1 it seems the error is actually coming from a miss-measured skeleton in the label data meaning improved network accuracy actually increases the measured error. The results plots (figures **??** and **??**) show that the label data, particularly the $Y[0]$ component, changes sharply at the points of greatest error which indicates a jump in the Kinect skeleton. Figure **??** shows the image and labelled skeleton for frame 257 of this sequence which is the position of the second peak in the loss graph. We see that the Kinect's algorithm has been confused by the occlusion of the left leg causing it to measure the left foot in front of the right which we see from the depth image is not the case. The network is not fooled by this occlusion, predicting a point near the maximum $Y[0]$, the skeleton nearest to this point shows the right foot correctly in front of the left. This was also the case around 213 and 303 where we have other peak in loss.

(a) Labelled skeleton.

(b) Depth image.

(c) Skeleton nearest to network's prediction.

Figure 20: Shows the image label and prediction for frame 257 of subject 1 normal 1. The results (figure **??**) showed a large error at this frame which is infact due to a mismeasured skeleton.

4.2 Network Errors Vs. Label Errors Vs. Image Errors

Erroneous skeletons in the labels for tested data are common. The largest losses across all the tested data occur at the beginning of sequences Subject 2 Normal 3 (results figure **??**, skeleton example figure **??**), Subject 3 Normal 1 (results figure **??**, skeleton example figure **??**), Subject 7 RL (results figure **??**, skeleton example figure **??**) and Subject 10 LL (results figure **??**, skeleton example figure **??**). As mentioned in section **??** skeleton data which is too dissimilar from a skeleton with neutral pose are discarded, which typically removes the frames at the beginning and end of the sequence where the subject is out of the optimum range of the sensor. However this cut can not be too stringent without excluding potential valid but abnormal poses from the data. Also poses that are well within this cut, but still far from the true pose are not excluded.

The examples displayed in **??**, **??** and **??** prove that the network can be accurate at distances greater than the Kinect algorithm, provided that the pose in the image is similar to some in the rest of the training data which do have accurate skeletons. This could be due to the difference in methods of the Kinect skeleton algorithm [**?**] and the network. We believe that the depth comparison features (presented in equation **??** and figure **??**) used for pixel body part classification in that work are are sensitive to range from the sensor. At increased range the values of these features must change since the width of body parts in

pixels will be smaller producing different values for the same body location and offsets at different ranges. Additionally, at longer range there will be fewer body pixels meaning that the statistics become noisier and any errors have a larger effect on the final joint position estimation. Although this is not discussed in their work, this would seem to make their trained random forests become inaccurate with range. In contrast our method scales the subject's body to a fixed size. At increased ranges we merely have less information due to the reduction in depth precision at longer ranges and due to pixelation on upscaling. These effects will likely cause a loss in precision, put poses should still be reasonably accurate provided there are similar poses represented in the training data.

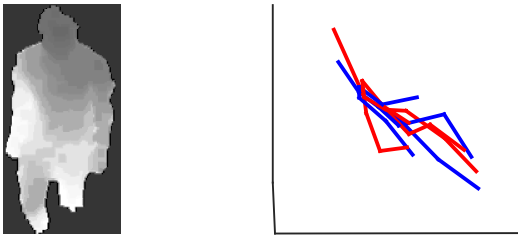
Whilst these erroneous skeletons are particularly common at the beginning of sequences they can also occur mid sequence when the Kinect algorithm is fooled by a challenging pose/image, particularly occlusions. This is the case in the Subject 1 Normal 1 sequence as discussed in section **??** with an example shown in figure **??**. Other examples are displayed figure **??**.

There are also frames with large error where the skeletons do appear accurate. Some examples of these are shown in figure **??**. These can occur where there are poses which receive a strange mapping when compared to similar images in the training data such as in figure **??** where a right leg forward pose receives a negative $Y[0]$, or figure **??** where a close to the camera lent in image, which normally receive large $Y[1]$, is instead mapped to a regular $Y[1]$.

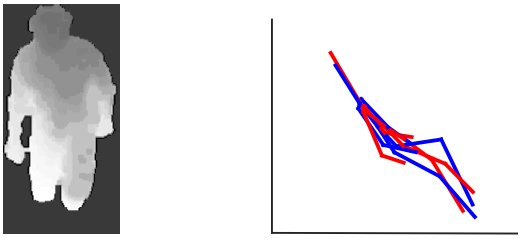
Another type of error is caused by bad images which were not removed during preprocessing. Typically these are when the back ground subtraction failed, the subject had not fully faced the sensor or the subject is too close to the sensor. They should have been removed during pre-processing but were missed. Examples are shown in figure **??**

In figure **??** we analyse every error further than the 1σ from the mean for all fine tuned sequences. Each of these 255 frames is assigned 1 of 5 classes: a network error (red), a skeleton error (green) or an image error (magenta).

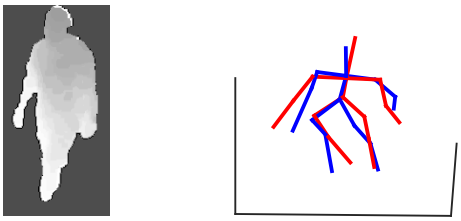
In each frame we studied the images and then examined the Kinect skeleton and compared the position of the prediction and label on the manifold. Only if the skeleton is clearly inaccurate is a skeleton error assigned. This has generally only been determined when one leg is measured in front of the other and the depth values of the image tell us otherwise or there is a clear double measurement of one foot as in figure **??**. In this case if the network's



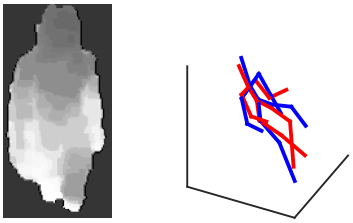
(a) Subject 2 Normal 3 frame 117 loss = 1.13. Labelled skeleton has legs at a similiar depth, network's prediction has right leg far in front of left.



(b) Subject 3 Normal 1 frame 162 loss = 1.30. Labelled skeleton has right leg in front of left, network measures the opposite which is correct.

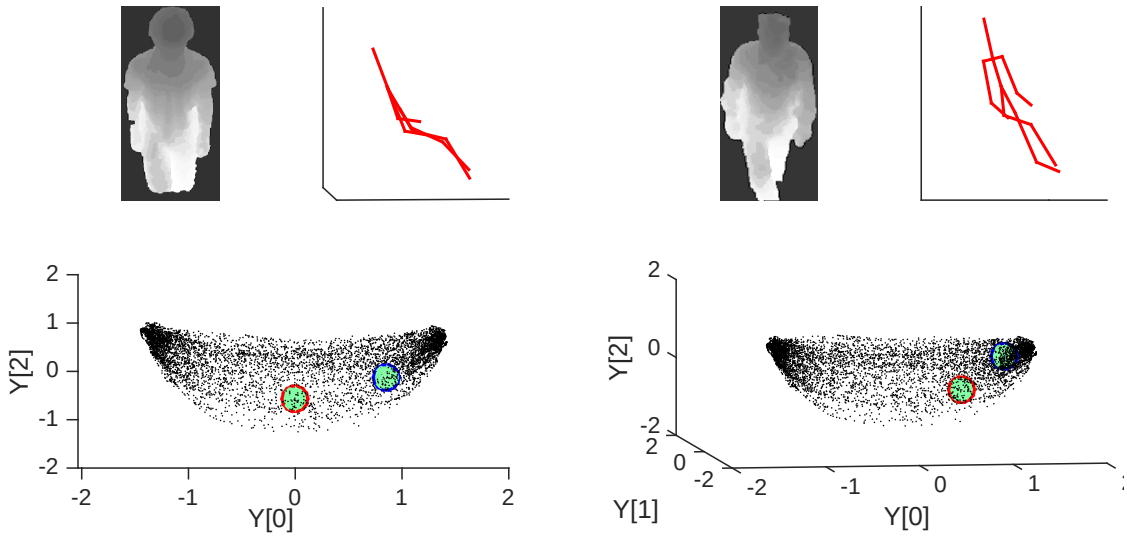


(c) Subject 7 RL frame 75 loss = 0.97. Labelled skeleton is not accurate. The network also fails since this is a strange pose where the subject appears to be turning on to the stairs. Since there are few examples of poses of this kind in the training data, and the ones that are have bad skeletons we can not expect better performance.



(d) Subject 10 LL frame 165 loss = 2.63. Labelled skeleton legs crossed over right above left, network measures left leg forward correctly.

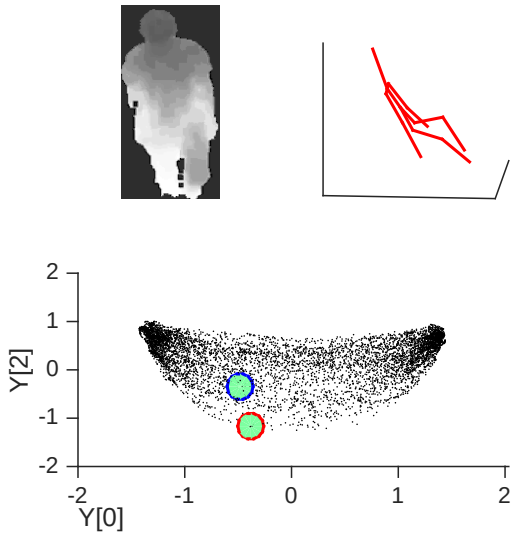
Figure 21: Out of range skeleton errors: shows labelled skeletons in red and the skeletons nearest to the network's predicted pose vector in blue for each of the images. These are some of the tested images which produced the greatest errors, occuring at the begining of sequences where the subject is outside of the Kinects optimum range. We find that these errors can be attributed to poor ground truth with the network able to produce a more accurate measure provided that similar poses with correct labels do exist in the training data.



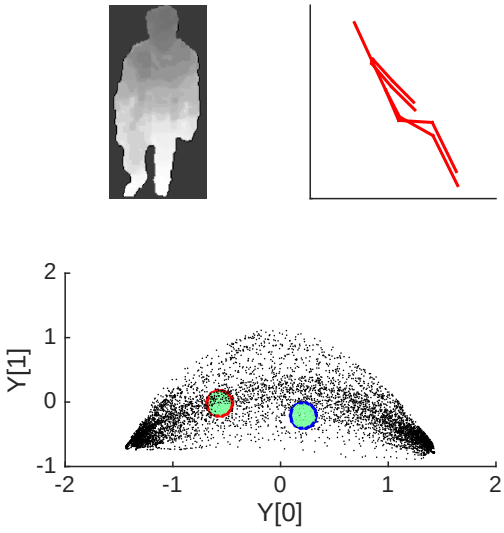
(a) Subject 2 Normal 3 frame 275. Loss = 0.5178. The skeleton of this frame is determined to be inaccurate since relative leg positions disagree considerably with the depth image.

(b) Subject 4 Normal 1 frame 205. Loss = 0.5738. The skeleton of this frame is determined to be inaccurate since there is a clear error in the left foot position being measured at the same position as the right due to occlusion.

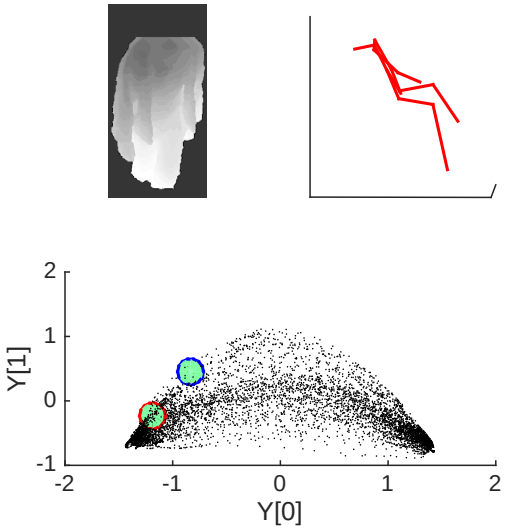
Figure 22: Skeleton errors: shows for each image the labelled skeleton and corresponding pose vector circled in red, and the network's prediction in blue.



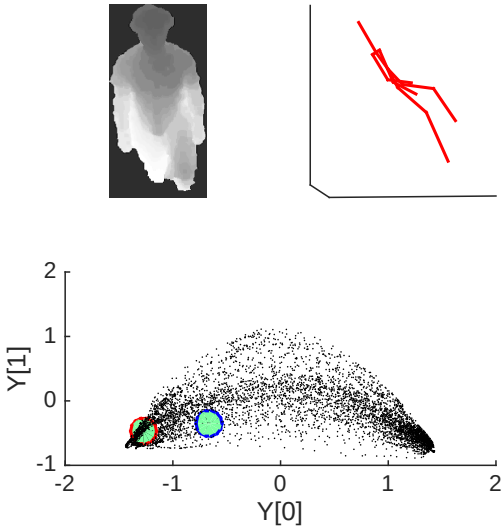
(a) Subject 2 Normal 3 frame 143 loss = 0.4268. The cause of errors in this sequence is discussed in section **??**.



(b) Subject 12 RL frame 167 loss = 0.3274.



(c) Subject 12 Normal 1 frame 253 loss = 0.3105.



(d) Subject 11 Stop x2 frame 311 loss = 0.2779.

Figure 23: Network Errors: shows label skeleton and corresponding pose vector in red and the network's predicted pose vector in blue for each of the images. In these cases the Kinect skeletons are deemed accurate. By design, the network predicts a pose vector typical of images similar to those it has seen in training. Network errors are caused by under represented pose vectors as in (a), or by inconsistencies with the training data as in (b) and (c).

(a) Subject 12 Normal 1 frame 239 loss = 1.06. A poor foreground mask slipped through the cuts due to its roughtly human shape and size.

(b) Subject 10 LL frame 444 loss = 0.77. Subject is too close to the sensor; legs are not visible.

Figure 24: Image Errors: shows labelled skeletons in red and the skeletons nearest to the network's predicted pose vector in blue for each of the images. These are some of the tested images which produced the greatest errors. These errors can be attributed to images which were meant to be removed during preprocessing, either because of bad masks or because the subject was too close to the camera or because they were not fully turned towards the camera.

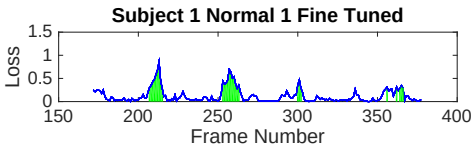
prediction in $Y[0]$ agrees with the image; i.e. right leg forwards positive $Y[0]$, left leg forwards negative $Y[0]$, legs at equal depth $Y[0] \sim 0$, then we deem the network accurate otherwise we determine that both network and skeletons are erroneous. We do not use the nearest skeleton method to determine network accuracy because it is not particularly robust; by nature of the skeleton to manifold mapping some noisy skeletons can exist close to other skeletons which would appear a reasonable prediction. We try to give an objective classification however this can be difficult; figure **??** presents some borderline cases.

If we exclude skeleton and image errors the mean error on the tested sequences drops to 0.0786, with a standard deviation of 0.0693.

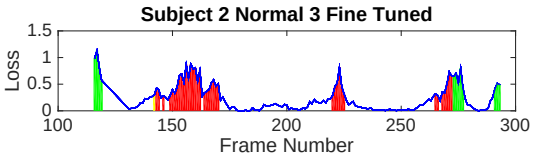
4.3 Movement Quality

[**?**] uses two measures: pose quality and dynamics quality for assessing the quality of human movement from the pose vectors of the stair ascent. These quantities were computed for us by the authors of [**?**] using the network predictions. These results are presented in figure **??**.

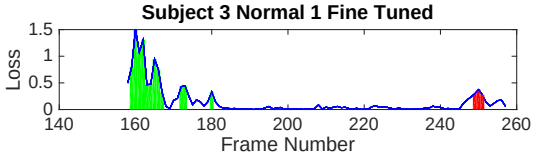
The pose quality is an estimate of the likelihood of a instantaneous pose vector being a normal pose. It is a probability density function of pose vector learnt form those normal poses



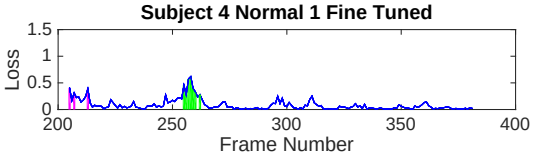
(a) adjusted mean loss = 0.0738 std = 0.0698.



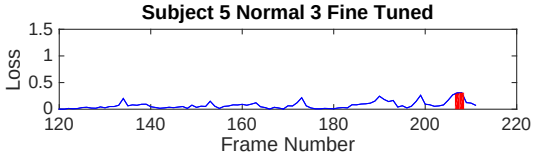
(b) adjusted mean loss = 0.1932 std = 0.2126.



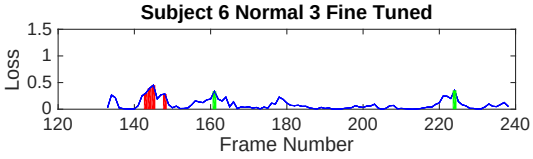
(c) adjusted mean loss = 0.0650 std = 0.0902.



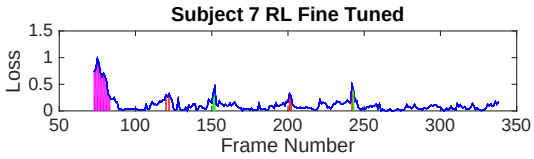
(d) adjusted mean loss = 0.0669 std = 0.0617.



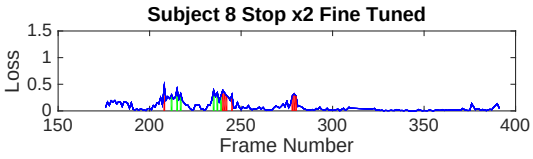
(e) adjusted mean loss = 0.1214 std = 0.1167.



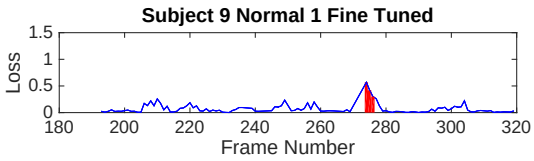
(f) adjusted mean loss = 0.0817 std = 0.0895.



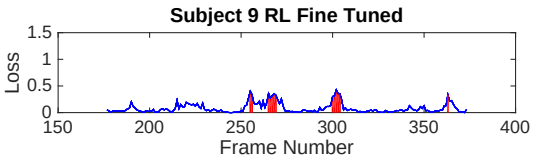
(g) adjusted mean loss = 0.0875 std = 0.0698.



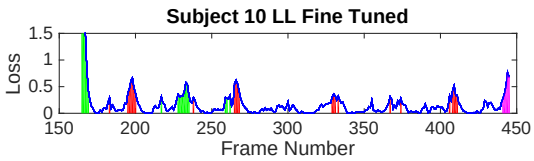
(h) adjusted mean loss = 0.1932 std = 0.2126.



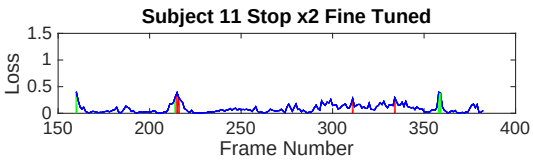
(i) adjusted mean loss = 0.0650 std = 0.0902.



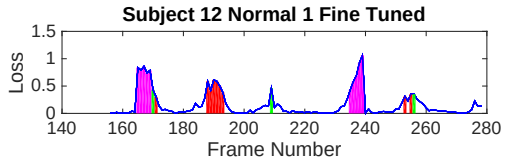
(j) adjusted mean loss = 0.0942 std = 0.1061.



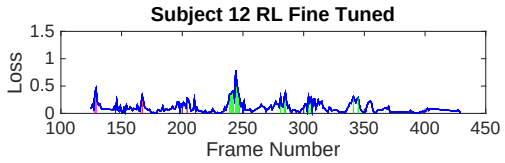
(k) adjusted mean loss = 0.0941 std = 0.1085.



(l) adjusted mean loss = 0.0786 std = 0.0693.



(m) adjusted mean loss = 0.0815 std = 0.0821.



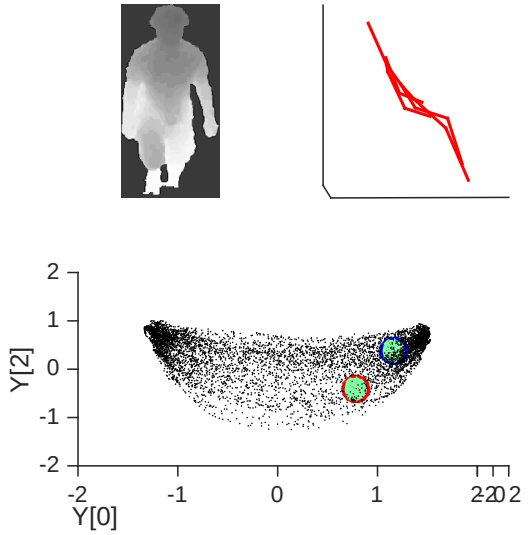
(n) adjusted mean loss = 0.0784 std = 0.0819.

Figure 23: Plots of the loss i.e. the distance between the labelled pose vector and the network's predicted pose vector for the subject fine tuned models. Every frame with loss greater than 1σ from the total mean is categorised as either a network error (red), an image error (magenta) or a skeleton error (green). Examples of each of these errors are shown in figures **??**, **??** and **??** respectively. The mean for each sequence is recomputed after excluding skeleton and image errors to give a more reliable estimate of the network's true precision. The collective mean after excluding these errors drops from 0.1565 to 0.0874.

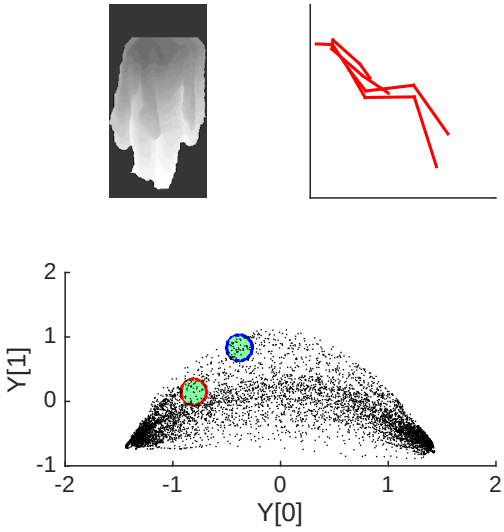
used to train the model. In this case these models are trained for the labels of the 17 normal sequences of subjects 1-6. For all sequences, normal and abnormal, the pose quality is generally expected to remain roughly within normal limits since the abnormality of these sequences is in the dynamics e.g. freezing of gait rather than in the pose. A threshold for abnormality was determined empirically at -5, this is shown in figures **??** as a dashed magenta line.

Generally the lower the loss is, the better the pose quality tracks that of the labels. There are however some regions around the edges of the manifold where small changes in pose vector produce large changes in quality. For example around frame 100 in Subject 7 RL (figure **??**) we find a discrepancy of the pose quality between labels and predictions of $\sim$-7 despite the loss being only $\sim$0.02. A similar case occurs in Subject 11 Stop x2 around frame 160. We believe that ideally such small changes in pose should not be producing such large differences in quality. Given a greater collection of normal pose training data the model may be more robust and this effect could be lessened. Alternatively some sort of smoothing may need to be applied to the model.

For the movement quality score each sequence is separated into gait cycles and each frame is assigned a coordinate equal to its position in the cycle. Again a probability density function on pose vector and its stage of the movement cycle is learnt from the normal sequences of



(a) Subject 11 Stop x2 frame 211 loss = 0.3881. A difficult case where the image and network's prediction seem plausible but the skeleton does too. In this case we give the skeleton the benefit of the doubt and assign a network error.



(b) Subject 12 Normal 1 frame 256 loss = 0.3511. Here the skeleton says the right knee is behind the left. Image seems to show them at equal depth so we assign a network error.

Figure 24: Shows some borderline cases when attributing the causes for errors in figure **??**. We try to be as objective as possible only ascribing a skeleton error if there is a clear error with the ground truth. These cases are determined to not be clear enough so we assign network errors in both cases.

subjects 1-6. This is displayed on a heat map in figure **??**. As with pose quality there are some areas, such as around $Y[0] = -1, X = 0.3$ the likelihood can fall sharply with $Y[0]$ which can be seen on this figure. The range of $Y[2]$ values which are deemed normal is much broader than that of $Y[0]$ or $Y[1]$ which means that the fact the network tends to measure this with lower accuracy should have less effect on the final results. Again an empirically determined threshold of -10 is used to define abnormality. Generally it is expected that normal sequences should remain above this threshold, left/right leg lead (L/RL) sequences should dip below every cycle and freeze (Stop $\times$ N) sequences should go below at every freeze of gait. However even for the labels there is a tendency for normal sequences to produce abnormal scores in places when stride times are not completely consistent as in the case of those presented in figure **??**. As with pose quality smaller loss in the networks predictions should produce a better agreement with label dynamics quality scores. However small mis-measurement can again produce large differences in the score.

In the R/LL sequences (figures **??** and **??**) we find that the network predictions tend to produce greater abnormality in the correct places than the labels did. For these sequences dynamics abnormality is increased considerably when the pose vector does not go beyond $Y[0] = 0$ for the frames where both legs are on the same step. When this occurs the path for the gait cycle enters the dark blue regions of the normality model for this component (shown in figure **??**). Looking at the sequences Subject 10 LL and Subject 7 RL (**??** and **??**) its seems that the network tends to measure $Y[0]$ below the label value at this point in the gait cycles. Comparing the predicted pose vectors, the images and the skeletons of frame 116 of Subject 7 RL, shown in figure **??**, it does seem that the Kinect has over-measured the position of the left knee. This leads it to produce a pose vector in the negative region of $Y[0]$ which in turn produces a normal dynamics quality score, where as the network's prediction is much closer to $Y[0] = 0$ producing an abnormal dynamics score. We find the same situation in the Subject 10 LL sequence with the right foot measured slightly in front of the left in these frames as shown in figure **??**.

However, the other tested L/RL sequence Subject 9 RL (figure **??**) exhibits the opposite behaviour at some of these cycle points i.e. predicts slightly negative $Y[0]$ when both feet should be on the same step. Looking at the images for these frames (frame 256 shown in figure **??**) a reason for this may be that the subject's left arm swings down across his knee which

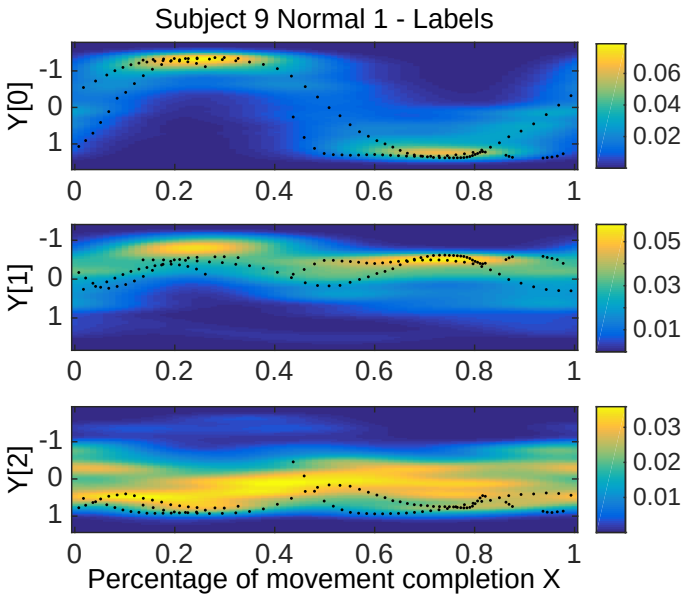
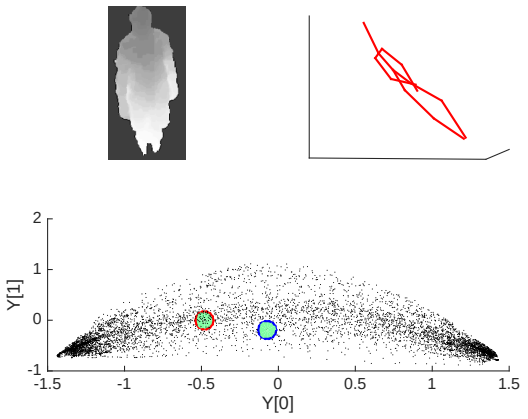


Figure 25: Shows the dynamics quality model (colour) as a function of pose vector and movement stage. The black dots represent frames from the Subject 9 Normal 1 sequence after its division into gait cycles. The strongest indication of normality is the value of Y[0]. Y[2] has a much broader range of normal values.

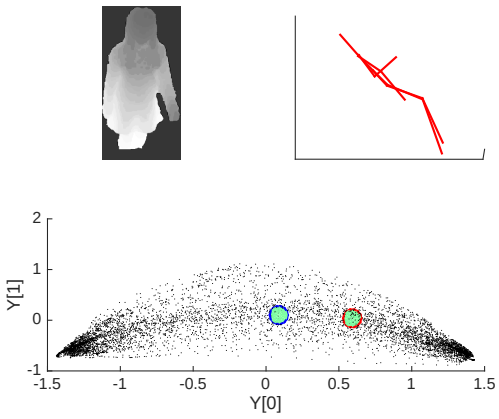
the network may be mistaking for a knee at increased depth. In the other points of this kind, where the network prediction's $Y[0]$ is equal or less than the skeleton's we find that the arm does not cover the knee as seen in figure **??**. This is an understandable error for the network to make. There were probably too few training examples where the arm covered the knee/hip in this manner for the network to learn to gauge this effect.

4.4 Processing Time and Memory Requirements.

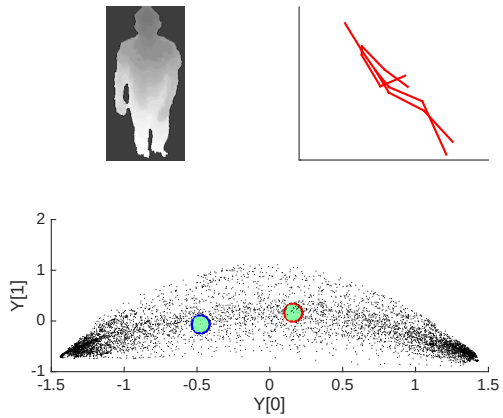
On the GeForce GTX 750 GPU the average forward pass time is 9.68827 ms. This is the time required to pass one image through the network and produce the pose vector. Even allowing some additional time for the image pre-processing on such hardware this system could operate in realtime at around 100 fps. On a single CPU the average forward pass time is 265.556 ms, a frame rate of around 4. This is comparable to the speed of Shotton et al.'s algorithm [**?**] which runs at ∼200 fps on the Xbox 360 GPU.



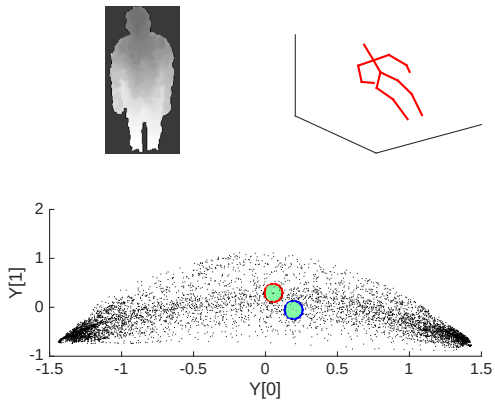
(a) Subject 7 RL frame 116. Kinect skeleton measures left knee slightly in front of right but image shows to them at equal depth.



(b) Subject 10 LL frame 365. Kinect skeleton measures right foot slightly in front of the left.



(c) Subject 9 RL frame 256. In this frame the network measurement is worse than the Kinect's. We think the presence of the subject's arm over the left knee causes the network to think the knee is raised.



(d) Subject 9 RL frame 188. In the same sequence when the arm is not covering the knee the network measurement is in line with the other R/LL sequences.

Figure 26: Shows the images, label skeletons and their corresponding points on the pose manifold (red) and the networks prediction (blue) for frames in the 3 tested L/RL sequneces where both feet are on the same step. In LL sequences we expect the pose vectors to remain in the negative $Y[0]$ region of the manifold and RL sequences the positive. We find the Kinect tends to over measure this phase of the movement which causes these sequences to not always be properly classified as abnormal. In the majority of cases the network measurements do a better job at identifying the abnormality.

A single image and its activations require 8.68 MB of memory. The network's weights require 227.5 MB.

5 Discussion

Due to inaccuracy of the Kinect skeletons there is not a completely consistent mapping between image and the pose vector.

5.1 Limitations

5.1.1 Abnormal Poses

5.1.2 Reconstruction of Full Pose

5.2 Directions of Future Work

6 Conclusion

Appendices

A Pre-existing SPHERE System For Movement Quality Analysis

In this section we will cover the existing system (as originally described in [**?**] and [**?**]) highlighting the points to be considered for our work.

As mentioned in section **??** the aim of the system is to quantify the quality of movement. This is achieved through comparing the recorded motion to a taught reference model of perfect motion. Although it has been evaluated specifically for gait measurements, the system aims to be widely applicable. Provided with suitable training data demonstrating perfect motion it could be applied to physiotherapy exercises, or in a sports movement optimisation application with little adaptation. This has been demonstrated with the system being applied to boxing

and sitting-standing motions presented on the SPHERE web page[3].

The gait analysis system has been trained using the SPHERE-staircase2014 dataset [**?**]. This dataset includes 48 sequences of 12 individuals walking up stairs, captured by a Kinect camera placed at the top of the stairs in a frontal and downward-looking position. It contains three types of abnormal gaits with lower-extremity musculoskeletal conditions, including freezing of gait and using a leading leg, left or right, in going up the stairs. All frames have been manually labelled as normal or abnormal by a qualified physiotherapist. 17 sequences of normal walking from 6 individuals were used for training purposes and 31 sequences from the remaining 6 subjects with both normal and abnormal walking were kept for testing of the system. An example with the skeleton overlaid is shown in **??**.

The system can be considered as a pipeline, as represented in **??**. Once deployed in real home environments , another software component being developed in SPHERE will be responsible for human detection and recognition, hence we can presume input depth images similar to those contained in the dataset, and also knowledge of the person. The skeleton tracking packages from section **??** extract the joint positions of the body in the image. The skeletons are averaged over a temporal window to smooth noise before being scaled, rotated and translated to normalise the pose (these steps will be detailed further in section^{insertreferencelater}).

Next the low-level feature extraction stage builds a feature vector out of the skeleton data to encode the pose. Tao et al. tests and compares a number of viable feature representations for the skeleton data. They find that using a vector of the each joint coordinate concatenated performs best overall [**?**].

This joint position vector is then processed using the non-linear dimensionality reduction method Diffusion Maps [**?**]. The stage is first trained offline, building the manifold representation from the training subset of the data. The characteristic of manifolds produced by Diffusion Maps is that the data retain the relative euclidean distances between point in the reduced space. This representation is then extended to new data in the testing phase by projecting new skeletons onto the existing manifold.

Next the reduced pose vectors $\mathbf{Y}$ of the training data is used to learn two probabilistic continuous HMM (Hidden Markov Model) models, one of instantaneous pose, and one of

[3] www.irc-sphere.ac.uk/work-package-2/movement-quality

dynamics. The normality of a pose is then the likelihood of the of $\mathbf{Y}$ being described by the pose model. Similarly the dynamical model computes a quality based on the likelihood of $\mathbf{Y}$ given the proceeding frames and the model of normal dynamics. Thresholds on these two normality scores are used to classify each frame normal or abnormal.

The system is 'online' in that it measures abnormality on a frame-by-frame basis, rather than processing a recorded sequence offline and measuring across the full sequence. This enables it to produce data on the parts of the motion that are deviating from normality, which is a benefit.

B A User Guide for Pre-Processing Scripts

The SPHERE staircase 2014 dataset and skeleton processing scripts of [**?**] can be downloaded from [**?**].

The pre-processing pipeline operates on the 3 main data arrays:

- low$_{l}evel_{f}eaturesalistofalltheselectedskeletons(i.e.theonesthatsurvivethesimilaritycut)$
- high$_{l}evel_{f}eaturesthecorrespondingmanifoldcoordinates$

line$_{n}umbersthistellsuswhichimagesfromeachsequenceofthedatasurvivedthecutandremaininthedataarraysal$

- Call prepare$_{m}odels.me.g.prepare_{m}odels(list_{n}ames2, [], 1, 9,' procrustes',' joint_{p}osition', 3, 1, 0.05, 1, 1,' ma$
  $6Wflip', 1)$

  where list$_{n}ames2isacellarraywitheachcellcontainingpathstothefilesstoredintheSPHEREstaircasedata$
  $/Shared_{d}ataset_{B}MVC2014_{s}tairs_{g}ait/stairs_{g}ait_{B}MVC2014_{s}keleton_{o}nly/training_{d}ata/full_{s}equences$

This saves a .mat file save('selected$_{s}keletons_{s}1-6Wflips',' low_{l}evel_{f}eatures',' low_{l}evel_{f}eatures_{f}lipped',' high_{l}ev$

and a manifold object from which you can call testingHighLvlFeat = manifold.projection(low$_{l}evel_{f}eatures(f, :$
$))$; $foreachoftheskeletonsofthetestingdatatogetthecorrespondinghighlevelfeaturespoint.$

B.1 Scripts

```
function prepare_models_FINAL(training_data_filenames,
    list_frames, smoothing, windowSize, alignment_type,
```

```
    feature_type, numDim, use_kNN, robust_coef, LBcoef, Nsteps,
    manifold_filename, periodic)
% called like: prepare_models_FINAL(list_names2,[],1,9,'
    procrustes','joint_position',3,1,0.05,1,1,'s',1)

%training_data_filenames : list of filenames
% list_frames : first and last frames to be used. Only used
    when Procrustes
%   alignment is not used.
%periodic : binary - 1 = periodic movement, 0 = non-periodic
%% arguments for preprocessor
% smoothing : 1 for smoothing of skeleton coordinates (
    recommended), 0 otherwise
% windowSize : size of window
% alignment_type : string that indicates the type of alignment
    to use for the
%   skeletons. Choices available: procrustes, torso_scaling,
    none.
% feature_type : string that indicates the type of features to
    extract from
%   the skeleton. Choices available: angles, joint_position,
%   joint_velocity, pairwise_relative_positions,
    pairwise_relative_angles
% the skeleton alignment types recommended per feature
    extraction type are:
% 'procrustes' + 'joint_position'
% 'procrustes' + 'joint_velocity'
% 'none + 'pairwise_relative_angles'
% 'torso_scaling' + 'pairwise_relative_positions'
%% arguments for manifold
```

```
22 % numDim : number of dimension of the manifold
23 % use_kNN : binary - 1 recommended
24 % robust_coef : weight of the robust extension - around 0.01 is
       fine
25 % LBcoef : 1 for Laplace-Beltrami
26 % Nsteps : 1 is fine
27 % manifold_filename : name under which the manifold should be
       saved
28 file_offsets =
       [102,111,174,171,88,81,141,106,185,321,113,230,134,120,212,140,133,218,66


29
30 nb_seq = length(training_data_filenames);
31
32 raw_data = [];
33 T = [];
34 low_level_features = [];
35 flipped_low_level_features = [];
36 file_names = [];
37 line_numbers = [];
38 isNormalSeqNonFlipped = [];
39 preprocessor = Preprocessing(smoothing, windowSize,
       alignment_type, feature_type,false);
40 preprocessor_flip = Preprocessing(smoothing, windowSize,
       alignment_type, feature_type, true);
41 image_numbers = [];
42 for s=1:nb_seq
43     % reading data
44     data = load(char(training_data_filenames(s)));
45     S = data(:,2:end);
```

```
46      P = (size(S,2))/4; % #joint points
47      idx = reshape(1:P*4,4,P);
48      index = reshape(idx(2:4,:),3*P,1);
49
50      r_data = S(1:end,index);
51      r_data_size = size(r_data,1);
52
53
54
55      % plot_skeletons(r_data);
56      % extract low-level features
57      [new_features_non_flipped,selection_non_flipped] =
            preprocessor.preprocess(r_data);
58
59
60      % select frames to be used to build the models (i.e. reject
            noisy outliers)
61      if strcmp(feature_type, 'joint_velocity') == 1
62          selection = seAClection(2:end);
63      end
64
65      if strcmp(alignment_type, 'procrustes') == 0
66          % At the moment, only Procrustes alignment provides a
                selection of valid frames. Other methods must have
                user defined selections.
67          frames = load(char(list_frames(s)));
68          selection = frames(1):frames(2);
69      end
70
71  %      shape = reshape(low_level_features(10,:),3,15)';
```

```
72 %      PlotSkeleton_SDK(shape);
73 %      pause;
74 %      shape = reshape(flipped_low_level_features(10,:),3,15)';
75 %      PlotSkeleton_SDK(shape);
76
77 %    raw_data = [raw_data; r_data(selection,:)];
78     % T = [T; data(selection,1)];
79       newLowLvlFeatNonFlipped = new_features_non_flipped(
             selection_non_flipped, : );
80       newLowLvlFeatFlipped = flipX_low_level_features(
             newLowLvlFeatNonFlipped);
81 %
82 %      figure(1);
83 %      shape = reshape(newLowLvlFeatNonFlipped(10,:),3,15)';
84 %      PlotSkeleton_SDK(shape);
85 %      hold on
86 %     %plot_skeletons(flipped_low_level_features);
87 %
88 %      shape = reshape(newLowLvlFeatFlipped(10,:),3,15)';
89 %      PlotSkeleton_SDK(shape);
90 %     % pause;
91 %      hold off
92
93
94       low_level_features = [low_level_features;
             newLowLvlFeatNonFlipped];
95       flipped_low_level_features = [flipped_low_level_features;
             newLowLvlFeatFlipped];
96
97     % plot_skeletons(low_level_features);
```

```
98      %file_names = [file_names; char(training_data_filenames(s))
            ];
99
100
101     all_lines = 1:length(selection_non_flipped);
102     selected_lines = all_lines(selection_non_flipped);
103
104     line_numbers{s} = selected_lines;
105
106     image_numbers= [image_numbers;line_numbers{s}(:)+
            file_offsets(s)-1];
107
108
109     isNormal = strfind( training_data_filenames(s), 'Normal');
110      if(~isempty(isNormal{1}))
111          disp(training_data_filenames(s));
112         seqFlag = logical(ones(size(newLowLvlFeatNonFlipped,1)
                ,1));
113      else
114          seqFlag = logical(zeros(size(newLowLvlFeatNonFlipped
                ,1),1));
115      end
116      isNormalSeqNonFlipped = [isNormalSeqNonFlipped; seqFlag];
117
118
119 %      plot_skeletons(r_data(frames(1):frames(2),:))
120 %      plot_skeletons(new_features(frames(1):frames(2),:))
121 end
122
```

```
123 %load('/home/sphere/gait_cnn/datasets/scripts/scripts/
        sphereComp/finalUsedImgLineNumbers.mat');
124
125 %low_level_features =low_level_features(allUsedImgLineNumbers
        ,:);
126 clear data
127 clear S
128 clear P
129 clear idx
130 clear index
131 %clear raw_data
132 clear preprocessor
133 %image_numbers=[image_numbers;image_numbers];
134 % build manifold
135 %Manifold = Manifold('/home/sphere/gait_cnn/datasets/scripts/
        scripts/Manifold_scritps/manifold.mat');
136 flippedAndUnflippedFeat = [low_level_features;
        flipped_low_level_features];
137 line_numbers1to6=line_numbers;
138 %save('sub1-6lowlvlfeat' ,'low_level_features','
        line_numbers1to6');
139 disp(size(low_level_features,1));
140 high_level_features_catted = build_manifold(
        flippedAndUnflippedFeat, numDim, use_kNN, robust_coef,
        LBcoef, Nsteps, manifold_filename);
141
142 high_level_features_flipped = high_level_features_catted(
        2793+1:end,:);
143 high_level_features = high_level_features_catted(1:2793,:);
144 training_data_filenames = training_data_filenames;
```

```
145 line_numbers = line_numbers;
146 low_level_features = low_level_features;
147 low_level_features_flipped = flipped_low_level_features;
148 %save('selected_skeletons_s1-6Wflips','low_level_features','
      low_level_features_flipped','high_level_features','
      high_level_features_flipped','training_data_filenames','
      line_numbers1to6')
149 %save('dynamics_model_train_data', 'image_numbers', '
      high_level_features_catted');
150
151 figure,
152 hold on
153 %tmp=1:length(isNormalSeqNonFlipped);
154 %tmp2 = tmp(isNormalSeqNonFlipped);
155 isNormalSeqNonFlipped = isNormalSeqNonFlipped == 1;
156 plot3(high_level_features(isNormalSeqNonFlipped,1),
      high_level_features(isNormalSeqNonFlipped,2),
      high_level_features(isNormalSeqNonFlipped,3),'--b.')
157 NisNormalSeqNonFlipped = ~isNormalSeqNonFlipped;
158 plot3(high_level_features(NisNormalSeqNonFlipped,1),
      high_level_features(NisNormalSeqNonFlipped,2),
      high_level_features(NisNormalSeqNonFlipped,3),'.')
159
160 % high_level_features_Flipped = build_manifold(
      flipped_low_level_features, numDim, use_kNN, robust_coef,
      LBcoef, Nsteps, manifold_filename);
161 % low_level_features_Flipped = flipped_low_level_features;
162 % save('selected_skeletons_Flipped','
      high_level_features_Flipped','low_level_features_Flipped');
163
```

```
164
165
166 disp('done');
167
168 clear low_level_features
169
170 % build dynamics model (part 1)
171 %[X_training, Y_training, indexes_training, nb_cycles] =
      find_cycles(high_level_features, T);
172 %save('training.mat','X_training','Y_training','
      indexes_training');
173
174 %disp('Now run the following python script:');
175 %sprintf('./pdf_estimations.py %d %d bw step',numDim,periodic)
176 %disp('(Requires the Python library scikit_learn)');
177 %disp('0.09 is a good guess for "bw" for the BMVC2014 training
      dataset')
178 %disp('"step" is the step size of the grid on which the pdf is
      pre-computed. A small value requires more RAM. 80 was used
      in our experiments in 3D.')
179
180 %disp('When done, run prepare_models_part2');
181
182 end
```

```
1 function prepare_models_FINAL(training_data_filenames,
      list_frames, smoothing, windowSize, alignment_type,
      feature_type, numDim, use_kNN, robust_coef, LBcoef, Nsteps,
      manifold_filename, periodic)
2 % called like: prepare_models_FINAL(list_names2,[],1,9,'
      procrustes','joint_position',3,1,0.05,1,1,'s',1)
```

```
3
4  %training_data_filenames : list of filenames
5  % list_frames : first and last frames to be used. Only used
       when Procrustes
6  %   alignment is not used.
7  %periodic : binary - 1 = periodic movement, 0 = non-periodic
8  %% arguments for preprocessor
9  % smoothing : 1 for smoothing of skeleton coordinates (
       recommended), 0 otherwise
10 % windowSize : size of window
11 % alignment_type : string that indicates the type of alignment
       to use for the
12 %   skeletons. Choices available: procrustes, torso_scaling,
       none.
13 % feature_type : string that indicates the type of features to
       extract from
14 %   the skeleton. Choices available: angles, joint_position,
15 %   joint_velocity, pairwise_relative_positions,
       pairwise_relative_angles
16 % the skeleton alignment types recommended per feature
       extraction type are:
17 % 'procrustes' + 'joint_position'
18 % 'procrustes' + 'joint_velocity'
19 % 'none + 'pairwise_relative_angles'
20 % 'torso_scaling' + 'pairwise_relative_positions'
21 %% arguments for manifold
22 % numDim : number of dimension of the manifold
23 % use_kNN : binary - 1 recommended
24 % robust_coef : weight of the robust extension - around 0.01 is
       fine
```

```
25 % LBcoef : 1 for Laplace-Beltrami
26 % Nsteps : 1 is fine
27 % manifold_filename : name under which the manifold should be
       saved
28 file_offsets =
       [102,111,174,171,88,81,141,106,185,321,113,230,134,120,212,140,133,218,66


29
30 nb_seq = length(training_data_filenames);
31
32 raw_data = [];
33 T = [];
34 low_level_features = [];
35 flipped_low_level_features = [];
36 file_names = [];
37 line_numbers = [];
38 isNormalSeqNonFlipped = [];
39 preprocessor = Preprocessing(smoothing, windowSize,
       alignment_type, feature_type,false);
40 preprocessor_flip = Preprocessing(smoothing, windowSize,
       alignment_type, feature_type, true);
41 image_numbers = [];
42 for s=1:nb_seq
43     % reading data
44     data = load(char(training_data_filenames(s)));
45     S = data(:,2:end);
46     P = (size(S,2))/4; % #joint points
47     idx = reshape(1:P*4,4,P);
48     index = reshape(idx(2:4,:),3*P,1);
49
```

```
50      r_data = S(1:end,index);
51      r_data_size = size(r_data,1);
52
53
54
55      % plot_skeletons(r_data);
56      % extract low-level features
57      [new_features_non_flipped,selection_non_flipped] =
            preprocessor.preprocess(r_data);
58
59
60      % select frames to be used to build the models (i.e. reject
            noisy outliers)
61      if strcmp(feature_type, 'joint_velocity') == 1
62          selection = seAClection(2:end);
63      end
64
65      if strcmp(alignment_type, 'procrustes') == 0
66          % At the moment, only Procrustes alignment provides a
                selection of valid frames. Other methods must have
                user defined selections.
67          frames = load(char(list_frames(s)));
68          selection = frames(1):frames(2);
69      end
70
71  %      shape = reshape(low_level_features(10,:),3,15)';
72  %      PlotSkeleton_SDK(shape);
73  %      pause;
74  %      shape = reshape(flipped_low_level_features(10,:),3,15)';
75  %      PlotSkeleton_SDK(shape);
```

```
76
77 %     raw_data = [raw_data; r_data(selection,:)];
78     % T = [T; data(selection,1)];
79       newLowLvlFeatNonFlipped = new_features_non_flipped(
             selection_non_flipped, :);
80       newLowLvlFeatFlipped = flipX_low_level_features(
             newLowLvlFeatNonFlipped);
81 %
82 %       figure(1);
83 %       shape = reshape(newLowLvlFeatNonFlipped(10,:),3,15)';
84 %       PlotSkeleton_SDK(shape);
85 %       hold on
86 %      %plot_skeletons(flipped_low_level_features);
87 %
88 %       shape = reshape(newLowLvlFeatFlipped(10,:),3,15)';
89 %       PlotSkeleton_SDK(shape);
90 %     % pause;
91 %       hold off
92
93
94       low_level_features = [low_level_features;
             newLowLvlFeatNonFlipped];
95       flipped_low_level_features = [flipped_low_level_features;
             newLowLvlFeatFlipped];
96
97     % plot_skeletons(low_level_features);
98       %file_names = [file_names; char(training_data_filenames(s))
             ];
99
100
```

```
101         all_lines = 1:length(selection_non_flipped);
102         selected_lines = all_lines(selection_non_flipped);
103
104         line_numbers{s} = selected_lines;
105
106         image_numbers= [image_numbers;line_numbers{s}(:)+
                file_offsets(s)-1];
107
108
109         isNormal = strfind( training_data_filenames(s), 'Normal');
110          if(~isempty(isNormal{1}))
111              disp(training_data_filenames(s));
112             seqFlag = logical(ones(size(newLowLvlFeatNonFlipped,1)
                    ,1));
113          else
114              seqFlag = logical(zeros(size(newLowLvlFeatNonFlipped
                    ,1),1));
115          end
116          isNormalSeqNonFlipped = [isNormalSeqNonFlipped; seqFlag];
117
118
119 %       plot_skeletons(r_data(frames(1):frames(2),:))
120 %       plot_skeletons(new_features(frames(1):frames(2),:))
121 end
122
123 %load('/home/sphere/gait_cnn/datasets/scripts/scripts/
        sphereComp/finalUsedImgLineNumbers.mat');
124
125 %low_level_features =low_level_features(allUsedImgLineNumbers
        ,:);
```

```
126 clear data
127 clear S
128 clear P
129 clear idx
130 clear index
131 %clear raw_data
132 clear preprocessor
133 %image_numbers=[image_numbers;image_numbers];
134 % build manifold
135 %Manifold = Manifold('/home/sphere/gait_cnn/datasets/scripts/
        scripts/Manifold_scritps/manifold.mat');
136 flippedAndUnflippedFeat = [low_level_features;
        flipped_low_level_features];
137 line_numbers1to6=line_numbers;
138 %save('sub1-6lowlvlfeat' ,'low_level_features','
        line_numbers1to6');
139 disp(size(low_level_features,1));
140 high_level_features_catted = build_manifold(
        flippedAndUnflippedFeat, numDim, use_kNN, robust_coef,
        LBcoef, Nsteps, manifold_filename);
141
142 high_level_features_flipped = high_level_features_catted(
        2793+1:end,:);
143 high_level_features = high_level_features_catted(1:2793,:);
144 training_data_filenames = training_data_filenames;
145 line_numbers = line_numbers;
146 low_level_features = low_level_features;
147 low_level_features_flipped = flipped_low_level_features;
148 %save('selected_skeletons_s1-6Wflips','low_level_features','
        low_level_features_flipped','high_level_features','
```

```
    high_level_features_flipped','training_data_filenames','
    line_numbers1to6')
149 %save('dynamics_model_train_data', 'image_numbers', '
    high_level_features_catted');
150
151 figure,
152 hold on
153 %tmp=1:length(isNormalSeqNonFlipped);
154 %tmp2 = tmp(isNormalSeqNonFlipped);
155 isNormalSeqNonFlipped = isNormalSeqNonFlipped == 1;
156 plot3(high_level_features(isNormalSeqNonFlipped,1),
    high_level_features(isNormalSeqNonFlipped,2),
    high_level_features(isNormalSeqNonFlipped,3),'--b.')
157 NisNormalSeqNonFlipped = ~isNormalSeqNonFlipped;
158 plot3(high_level_features(NisNormalSeqNonFlipped,1),
    high_level_features(NisNormalSeqNonFlipped,2),
    high_level_features(NisNormalSeqNonFlipped,3),'.')
159
160 % high_level_features_Flipped = build_manifold(
    flipped_low_level_features, numDim, use_kNN, robust_coef,
    LBcoef, Nsteps, manifold_filename);
161 % low_level_features_Flipped = flipped_low_level_features;
162 % save('selected_skeletons_Flipped','
    high_level_features_Flipped','low_level_features_Flipped');
163
164
165
166 disp('done');
167
168 clear low_level_features
```

```
169
170 % build dynamics model (part 1)
171 %[X_training, Y_training, indexes_training, nb_cycles] =
      find_cycles(high_level_features, T);
172 %save('training.mat','X_training','Y_training','
      indexes_training');
173
174 %disp('Now run the following python script:');
175 %sprintf('./pdf_estimations.py %d %d bw step',numDim,periodic)
176 %disp('(Requires the Python library scikit_learn)');
177 %disp('0.09 is a good guess for "bw" for the BMVC2014 training
      dataset')
178 %disp('"step" is the step size of the grid on which the pdf is
      pre-computed. A small value requires more RAM. 80 was used
      in our experiments in 3D.')
179
180 %disp('When done, run prepare_models_part2');
181
182 end
```

C Networks

C.1 AlexNet