Masterarbeit im Fach Informatik
Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Computer Vision Group
Prof. Dr. J. Gall

Using 2D Pose Information for 3D Pose Estimation in Deep Networks

29th November 2016

vorgelegt von:
Muhammad Omer Saeed
Matrikelnummer 2776827

Gutachter:
Prof. Dr. J. Gall
Jun.-Prof. Dr. A. Yao

Betreuer:
M. Sc. Umar. Iqbal

Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Textauszüge und Grafiken, die sinngemäß oder wörtlich aus veröffentlichten Schriften entnommen wurden, sind durch Referenzen gekennzeichnet.

Bonn, 01. Januar 2016

Muhammad Omer Saeed

Abstract

3D human pose estimation is inherently a challenging task. While there have been success in 2D human pose estimation using data driven models such as convolutional neural networks (CovNets) and probabilistic models, the estimation of 3D human pose from a single RGB image remains challenging. This is mainly due to the many ambiguities inherent to monocular 3D reconstruction including occlusion and loss of depth information caused by the projection from 3D to 2D. The 3D pose estimation is also hard because of the many complex environments and backgrounds as well as varied type of colorful clothing used by humans in everyday life.

We address the problem of 3D human pose estimation from a single viewpoint image. This is an important and fundamental problem in many computer vision applications including human computer interaction, security and surveillance monitoring, image understanding, motion capture and action recognition in sequences. We test different convolutional neural network architectures with different training strategies to investigate which configuration is optimal for the task of 3D human pose estimation. We specifically also test how well can 2D pose information help us generalize and better train convolutional neural networks for 3D pose estimation. We evaluate our results on the Human3.6M dataset and compare our configurations with state of the art works on 3D human pose estimation.

Contents

1 Introduction

Over the last decade significant progress has been made in the area of human pose estimation. This progress has been made possible by the availability of large amount of annotated data which can be used for training, for instance regression models. Public availability of datasets including the Human3.6M [Catalin Ionescu & Sminchisescu 14] and the HumanEva [Sigal & Balan$^+$ 10] has greatly contributed to this task. Although these large scale datasets are recorded in restricted environments and thus represent only a small subset of the huge space of 3D human poses, the novel ideas of [Yasin & Iqbal$^+$ 16] as well as [Rogez & Schmid 16] has proposed ways of human pose prediction in general environments. The former proposed a dual source approach where one source consists of images with annotated 2D poses and the second independent source consists of accurate 3D motion capture data. During inference the 2D pose is estimated from the image and the nearest 3D poses are retrieved from the motion capture data which are then further refined. The latter proposes to synthesize large number of in the wild images which provide corresponding accurate 3D pose annotations thus theoretically providing infinite training data. The launch of Microsoft Kinect depth sensor cameras simulated a lot of research in this field using depth information in early 2010. Plenty of recent academic work use depth data for 3D pose estimation. [Ross Girshick 11] used depth information for reliable pose estimation using regression forest algorithm. However, current depth sensors are limited to indoor environments and cannot be used in unconstrained scenarios.

Even though the recent research has greatly pushed the results on the task of 3D human pose estimation, it is still considered largely unfinished. There are many subproblems of the 3D human pose estimation that have either not been solved yet or are partially solved. Some of these problems include partial occlusions, variability of human body and shape, high dimensional space of 3D human poses and loss of depth information. The task of 2D pose estimation on the other hand has achieved remarkable results especially after the introduction of deep learning to this task. Convolutional pose machines [Shih-En Wei & Sheikh 16] (see section 2.3) and Deepcut [Pishchulin & Insafutdinov$^+$ 16] have used convolutional neural networks to achieve state of the art results on 2D human pose estimation.

In this thesis we investigate the effect of using 2D pose information as a prior and a regularization for training end to end 3D human pose regression networks. We

test several approaches including predicting the 2D pose from the image on the fly using the convolutional pose machine [Shih-En Wei & Sheikh 16] and then feeding this information to the network fine tuned for the task of 3D pose estimation. We also test networks with loss for both 2D and 3D pose to investigate whether the features jointly learned for 2D and 3D pose estimation better generalize for the task of 3D pose estimation. We report our results on the Human3.6M dataset and compare the results with the state of the art.

1.1 Problem Statement

The task of human pose estimation can be formulated as the problem of localizing human joint coordinates in either 2D or 3D. This task is often part of a larger problem which mostly includes detecting subject(s) in the images and/or segmenting the human body parts. Recent deep learning architectures can handle these tasks quite well. Formally, we are interested in finding a posterior distribution of the pose parameters that best explains the current pose, given the current frame.

Let $y_t^i \in \mathbb{R}^N$ define the 3D human pose paramters where $y_t^i \in \mathbb{R}^N$ represents an N-dimensional vector that holds each body part's position, orientation and scale at frame t and $i = 1...J$ where J is the number of human body joints under consideration. Let also $x_t \in \mathbb{R}^{W \times H}$ be the current image frame at time t with width W and height H and three RGB color channels. Then the posterior distribution that we are interested in finding that best explains the current pose parameters is defined as,

$$\prod_{i=1}^{J} p(y_i^t | x_t)$$

1.2 Related Work

Human pose estimation has been extensively studied in recent literature because it is a fundamental problem in various computer vision applications including understanding human interactions, action recognition, video surveillance and security. Human pose estimation approaches can be classified in to two main categories: model based generative methods and discriminative methods. The pictorial structure model (PSM) [Felzenszwalb & Huttenlocher 05] [Eichner & Marin-Jimenez$^+$ 12] falls into the category of generative models and it is one of the most popular early methods used for 2D human pose estimation. PSM represents joints as vertices and joint relations as edges in a non-circular graph. Pose estimation then boils

down to inference on the graph which can be solved via optimization algorithms. This method has been successful on images where all the limbs of the person are visible, but is prone to characteristic errors such as double counting image evidence. The same approach was extended to 3D human pose estimation [Belagiannis & Amin$^+$ 14] [Burenius & Sullivan$^+$ 13] but for the 3D case the graph is complex and thus the inference was slow. [Belagiannis & Wang$^+$ 14] proposed a PSM based model for 3D pose estimation with polynomial complexity but as with other tree structured methods it suffered from the problem of confusing the joints of right and left limbs. To overcome this, there has been effort focused on coming up with powerful models, particularly by solving this left-right confusion [Jiang & Martin 08] [Cherian & Mairal$^+$ 14] by formulating the problem as a non-tree structure. These methods achieved better results by using Branch and Bound (BB) [Marinescu & Dechter 09] to search for the globally optimal solution. Such approaches were based on hand-crafted features (e.g., HOG and SIFT), and they were hence sensitive to noise and limited by the representational ability of the features. The 3D PSM proposed by [Kostrikov & Gall 14] combined discriminative and generative methods by training regression forests to estimate the probabilites of 3D joint locations and then inferring the final 3D pose by using the PSM. The inference still required a few minutes per frame.

Discriminative methods consider the task of pose estimation as a regression problem. The human pose estimation problem has seen a breakthrough since the introduction of deep learning to this task by the pioneer deep pose work [Toshev & Szegedy 13]. Deep pose cascades 3 stages of neural networks for estimating 2D human pose from monocular RGB images. The cascading process assumes that local evidence can be used to make accurate joint predictions although this is not true for the 3D case where context around the target joint is of immense importance for accurate predictions. [Shih-En Wei & Sheikh 16] show that using heat maps as intermediate supervision in a fully convolutional neural network can greatly improve 2D part detection. The work of [Sungheon Park & Kwak 16] is close to our thesis where they jointly learn features for 2D and 3D pose estimation in a network with multiple losses and argue that the jointly learnt features better generalize the network. [Bugra Tekin & Fua 16] account for the human pose structure by training an auto-encoder to project body joint positions to a high dimensional space and then learn a CovNet based mapping from the input image to this high dimensional space where the pose parameters are then predicted. [Li & Chan 14] propose a two way training strategy procedure to train a deep convolutional network for 3D pose estimation. Firstly, training is done for regression task to estimate joint positions relative to the root and secondly the training is done for the detection task that classifies each joint according to the local window evidence. [Bogo & Kanazawa$^+$ 16] combines deepcut [Pishchulin & Insafutdinov$^+$ 16] which is a discriminative method

using Covnets to estimate 2D joints with SMPL [Loper & Mahmood$^+$ 15] which is a 3D generative model to estimate 3D pose and achieve state of the art results on the Human3.6m dataset [Catalin Ionescu & Sminchisescu 14]. After the success of very deep residual networks [He & Zhang$^+$ 15] which won the ImageNet [Russakovsky & Deng$^+$ 14] 2015 classification challenge, residual networks have been used for solving many tasks in computer vision. [Xingyi Zhou & Wei 16] trained a 50-layer residual network for end to end 3D human pose regression which was pre-trained on ImageNet to achieve state of the art results on the Human3.6M dataset [Catalin Ionescu & Sminchisescu 14]. [Yasin & Iqbal$^+$ 16] take a non-parametric approach in which the detected 2D joints are used to look up the nearest 3D poses in a mocap dataset but since this 3D pose is retrieved on the go and the training is not done in an end to end fashion, this method is error prone although it is well suited to estimate 3D human pose in general complex environments.

Using the 2D pose information, undesirable 3D joint positions which generate unnatural human pose can be discarded. Therefore, 2D pose information can help to improve 3D pose predictions. Recent advances in deep learning are producing accurate methods for 2D pose estimation [Pishchulin & Insafutdinov$^+$ 16] [Shih-En Wei & Sheikh 16]. We use convolutional pose machines [Shih-En Wei & Sheikh 16] which give remarkably good 2D part heat maps to investigate the practical effect of using 2D pose information to directly regress the 3D human pose.

1.3 Overview

The thesis is structured as follows. **Chapter 2** contains a brief introduction of neural networks and convolutional neural networks with focus on different training techniques for optimal training of the networks. It also introduces the reader to the convolutional pose machines (CPM) and residual networks. **Chapter 3** explains the methods and approaches we used for different experiments and the motivation behind them. An introduction to the publicly available Human3.6M dataset is also provided in this chapter. **Chapter 4** discusses the evaluation protocol and the results of the experiments. The comparison with the state of the art is also given. **Chapter 5** provides a summary and a conclusion with possible future directions of work.

2 Background

2.1 Basics of Neural Networks

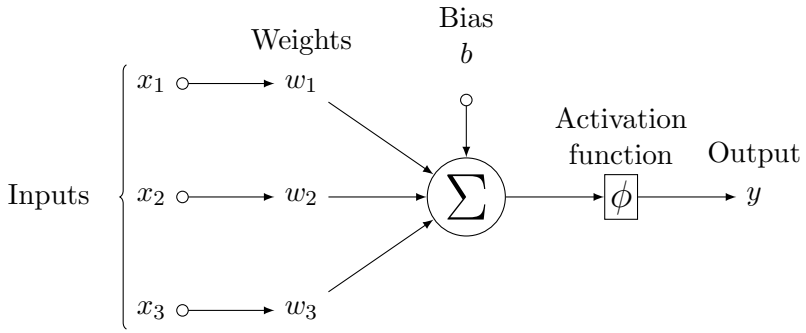

Figure 2.1: The Figure shows a single neuron with three inputs.

Neural networks are biologically inspired computational models. They are built from the basic units called neurons which are mathematical functions modeling the biological neurons. Each neuron receives one or more inputs and it produces an output which is the weighted sum of the inputs. The sum is passed through an activation function also known as the transfer function. The activation function is crucial for defining the behavior of the neuron and it is the main differentiating factor among different neurons. In practice, the neurons are placed in layers so that the output of one layer usually propagates to the input of the next layer and thus creating a neural network.

Mathematically, for a given neuron, let there be $n+1$ inputs with signals x_0 through x_n and weights w_0 through w_n. Then the output of the kth neuron is given by,

$$y_k = \phi \ (\sum_{i=0}^{n} w_{ki} \ x_i)$$

where ϕ is the activation function.

Neural networks are data driven i.e, they can learn weights via training using example data. The training can be done under supervised as well as unsupervised conditions.

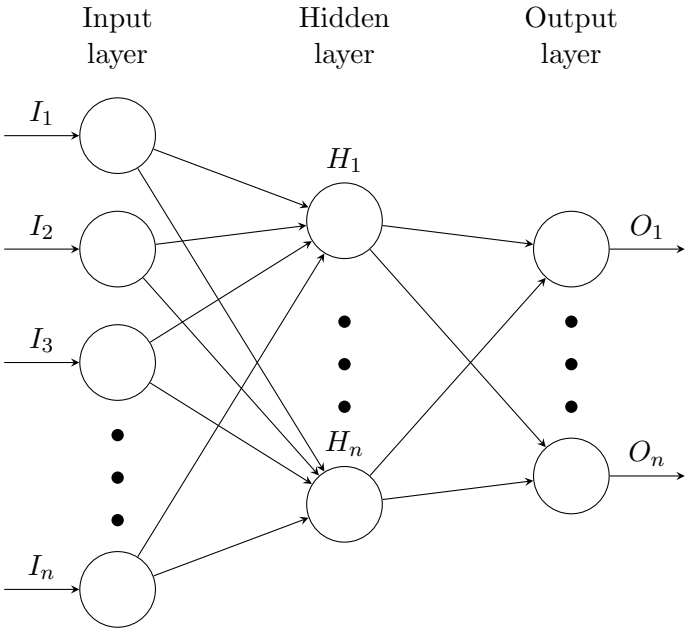

Figure 2.2: A Feed Forward neural network. The middle layer is called a *hidden* layer since the neurons in this layer are neither inputs nor outputs.

2.1.1 Activation Functions

For a multi-layer network it is important and advantageous that the activation function is non-linear [Hornik 88] since any multi-layer neural network with linear activations can be represented as a single layer network. Theoretically any differentiable function can be used as the activation function but in practice mostly logistic functions are used since they are easy to differentiate. Some of the common activation functions used are sigmoid, tan hyperbolic and rectified linear units [Nair & Hinton 10]. Each of them are briefly described in the sections below.

Sigmoid Function

A sigmoid function is a bounded differentiable real function that is defined for all real function values and has a positive derivative at each point [Han & Moraga 95]. It is defined by the mathematical formula,

$$\text{Sigmoid } (x) \ = \frac{1}{1+\exp(-x)}$$

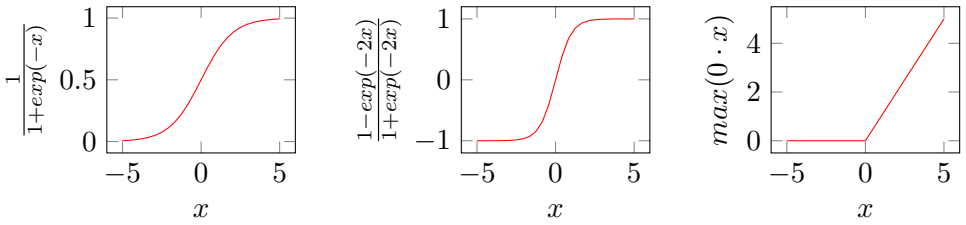

Figure 2.3: Left: Sigmoid function, the input value is squashed in to the range $[0, 1]$. Middle: Tan hyperbolic function, the input value is squashed between $[-1, 1]$. The tan hyperbolic function is preferred over sigmoid. Right: The ReLu function, the output is zero for input < 0 and linear with slope 1 for input > 0.

The function is plotted in Figure 2.1.1. The sigmoid function is strictly increasing and it squashes the input to the range $[0, 1]$ by converging very high negative values to 0 and very high positive values to 1. Over the years sigmoid activation function has decreased in popularity mainly because of the following two reasons:

- Sigmoid(x) is very small for x outside of $x \in [-5, 5]$. If x lies outside this range then the neuron 'dies' since any change to the weights is multiplied by Sigmoid(x) the weights of the neuron won't change and thus the value of the neuron remains fixed.
- Since Sigmoid(x) is always positive, all the weights will either move in the positive direction or the negative direction during a given step of gradient descent. This can cause jittering for updates of weights during training with small batch sizes.

Tangent Hyperbolic Function

The tan hyperbolic function is defined as follows,

$$Tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$$

The function is plotted in Figure 2.1.1. The tan hyperbolic function squashes the input between $[-1, 1]$ and since its not always positive, it doesn't have the problem with the weights moving to the same direction.

Rectified Linear Unit

The ReLu function is defined as follows,

$$ReLu(x) = max(0, x)$$

The function is plotted in Figure 2.1.1. ReLu was first introduced as an activation function by [Nair & Hinton 10] getting motivation from biology. ReLu provides sparse activation, it is cheap to compute because there is no exponential computation and also provides efficient gradient propagation since there are no vanishing or exploding gradient problems. This activation function is also scale invariant since $max(0, ax) = a\ max(0, x)$.

One disadvantage of ReLu is that if x is too small then this activation function 'dies' i.e, it does not produce any gradient.

2.1.2 Objective Functions

An objective function is a representation of the cost associated with the problem at hand at a certain stage. An optimization technique will seek to minimize this cost. Different optimization strategies are discussed in the following sections. Different objective functions focus on different attributes of the data and can thus be used in accordance with what is to be achieved with the problem at hand. In this section two standard objective functions are discussed, namely the least absolute deviations $L1$ and the least squared errors $L2$.

Least Absolute Deviations (L1)

$L1$ objective function calculates the absolute differences between the predicted values and the ground truth values. $L1$ can be formulated as follows,

$$L1_{obj} = \sum_{i=0}^{n} |y_i - h(x_i)|$$

where, y_i is the ground truth value, $h(x_i)$ is the predicted value and x_i denotes the feature set of a single sample. This objective function is robust to outliers.

Least Squared Errors (L2)

$L2$ calculates the squared difference between the ground truth and the predicted values. It can be formulated as follows,

$$L2_{obj} = \sum_{i=0}^{n} (y_i - h(x_i))^2$$

$L2$ objective function is highly sensitive to outliers in the dataset since the original difference is squared. Thus $L1$ objective function is preferred in the settings where outliers are known to exist in the dataset.

2.1.3 Regularizations

Overfitting is a major problem for predictive analysis and especially for neural networks. In the context of neural networks, overfitting occurs when the network begins to memorize the training data rather than learning to generalize from trend. This usually happens when the network is complex and has enough parameters to learn to perfectly predict the training data while the test time accuracy decreases since there is no generalization in the trained model.

Regularization is a method for combating overfitting. It helps to decrease the capacity of the model by usually introducing some additional constraints in the objective function. Although there are many ways to regularize, L1 regularization, L2 regularization, max norm and dropout are discussed below.

L1 regularization

L1 regularization is defined as follows,

$$R_{L1}(w) = ||w||_1 = \sum_{i=1}^{n} |w_i|$$

Thus our objective function becomes,

$$L_{obj+R_{L1}} = L_{obj} + \lambda R_{L1}(w)$$

The regularization term is scaled by a factor λ which is called the *regularization parameter*. The relative importance of the two terms of the objective function depend on the value of λ. When λ is small the first term of the objective function is given preference to minimize and when λ is large the second term is preferred to be minimized which effectively means preferring smaller weights.

L1 regularization also leads the weight vectors to become sparse which increases network performance. Neurons with L1 regularization uses only a small subset of important inputs and is resistant to noise.

L2 regularization

L2 regularization is defined as follows,

$$R_{L2}(w) = ||w||_2^2 = \sum_{i=1}^{n} w_i^2$$

Thus our objective function becomes,

$$L_{obj+R_{L2}} = L_{obj} + \lambda R_{L2}(w)$$

where λ is the regularization parameter as described before. L2 regularization has an intuitive property of heavily penalizing "peaky" weights and preferring diffuse weight vectors. This encourages the network to focus on all inputs rather than a few of them. L2 regularization is also referred to as *weight decay*.

Max Norm Constraint

Max norm constraint [Lee & Recht$^+$ 10] is an absolute upper bound on the magnitude of the incoming weight vector for every neuron. The neural network is optimized under the constraint $||w||_2 \leq c$. The constraint is imposed by projecting w onto a surface ball of radius c which makes sure that the maximum value any given weight can take is c. The constant c is a tunable hyper-parameter which can be determined using a validation set.

Dropout

Dropout [Srivastava & Hinton$^+$ 14] is a different kind of method to prevent overfitting. In the case of L1 and L2 the objective function is modified but in the case of dropout the model itself is modified. Dropout is implemented with only keeping a neuron active with some probability p and setting it to *zero* otherwise (see Figure 2.4). This forces the network to be accurate even with the absence of certain information. It can also be thought of as a way of approximately combining exponentially many different neural network configurations efficiently.

2.1.4 Optimization Techniques

Optimization is responsible for finding the best weights of the neural network to minimize the objective function. Currently, the most widely used optimization techniques are gradient based using back propagation although global search techniques

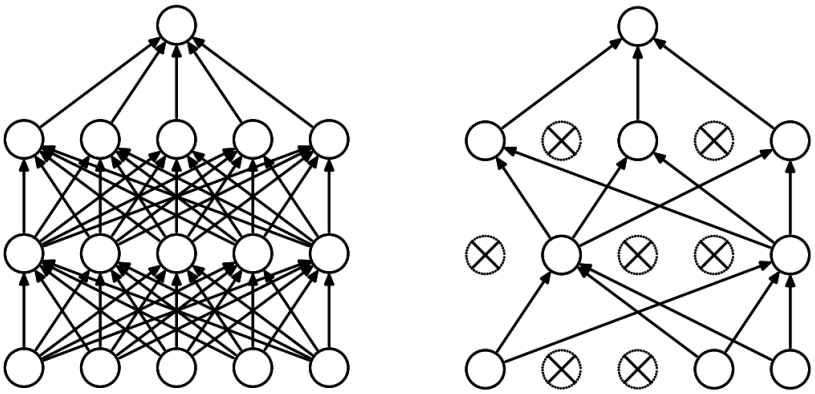

Figure 2.4: Left: Standard neural network, Right: After applying dropout. Taken from [Srivastava & Hinton$^+$ 14]

can also be used such as simulated annealing and genetic algorithm [R. Ghosh & Bagirov 05]. Here only the gradient based techniques are described since they are the ones used in our experiments.

Stochastic Gradient Descent (SGD)

Stochastic gradient descent is a gradient based learning algorithm. Generally each parameter update in SGD is computed with respect to a few training examples or a mini-batch. This reduces the variance in the parameter update and can lead to a more stable convergence. SGD updates the weights W by a linear combination of the negative gradient of the objective function $\nabla f_{obj}(W)$ and the previous weight update V_t. The learning rate α is the weight of the negative gradient and the momentum μ is the weight of the previous update. The update step is given by,

$$V_{t+1} = \mu V_t - \alpha \nabla f_{obj}(W_t)$$

$$W_{t+1} = W_t + V_{t+1}$$

The parameters α and μ are hyper-parameters and requires some tuning for best results. For further reading on SGD please refer to [Bottou 12].

Adam

Adam proposed in [Kingma & Ba 14] is a gradient based optimization method that computes adaptive learning rates for each parameter. The algorithm updates

exponential moving averages of the gradient m_t and the squared gradient v_t. It is the generalization of the AdaGrad method [John Duchi & Singer 11]. The update step is given by,

$$(m_t)_i = \beta_1(m_{t-1})_i + (1-\beta_1)(\nabla f_{obj}(W_t))_i$$

$$(v_t)_i = \beta_2(v_{t-1})_i + (1-\beta_2)(\nabla f_{obj}(W_t))_i^2$$

and finally,

$$(W_{t+1})_i = (W_t)_i - \alpha \frac{\sqrt{1-(\beta_2)_i^t}}{1-(\beta_1)_i^t} \frac{(m_t)_i}{\sqrt{(v_t)_i + \varepsilon}}$$

where α is the learning rate and β_1, β_2 and ε are hyper-parameters which can be tuned for best results. [Kingma & Ba 14] proposes to use $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

2.1.5 Training Techniques

Batch Normalization

During training the neural network, the distribution of each layer's input changes as the parameters of the previous layer change which can slow down the learning. This phenomenon is referred to as internal covariate shift and it can be addressed by normalizing the inputs of the layer by using batch normalization [Ioffe & Szegedy 15].

It is beneficial to use a batch of examples for training rather than one example at a time because it improves the gradient estimate over the example set and the training can be highly parallelized on modern hardware. Let $\mathcal{B}$ be a mini batch of m input examples $\mathcal{B} = \{x_{1...m}\}$, to apply batch normalization firstly the mean of the mini batch is calculated,

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$$

and then the variance of the mini batch is calculated as,

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

using $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$, the input can be normalized as follows,

$$\hat{x} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}}}$$

Simply normalizing each input of a layer can change what a layer can represent. For example, normalizing the input of a sigmoid function would cause the output to be bound to the linear region only. Thus it has to be made sure that the transformation inserted in the network can represent the identity transform. For this to be accomplished $\hat{x}$ is transformed to,

$$y_i \leftarrow \gamma \hat{x_i} + \beta$$

where γ and β are learn-able parameters which help to restore the representation power of the network.

Some further advantages of using batch normalization during training are as follows,

- It regularizes the model and reduces the need for dropout and other regularization techniques.
- It reduces the dependence of gradients on the scale of the parameters or their initial values.
- It allows use of higher learning rates during training.

2.2 Convolutional Neural Networks

Convolutional neural networks (CovNets) were first introduced by [LeCun Yan & Denker 89] for the task of handwritten number recognition. They are $feed\text{-}forward$ networks meaning that the connections between the units do not form a cycle. The connectivity pattern in CovNets is inspired by the organization of the animal visual cortex. From the work of [Hubel & Wiesel 68] on animal visual cortex it was found that cells in visual cortex are sensitive to small sub-regions of the visual field called receptive field. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images. Hence, CovNets are heavily used in the tasks of visual recognition.

2.2.1 Basic Layers Of CovNets

A CovNet can be built by stacking up different combinations of distinct basic layers. Some of the main layers are described below.

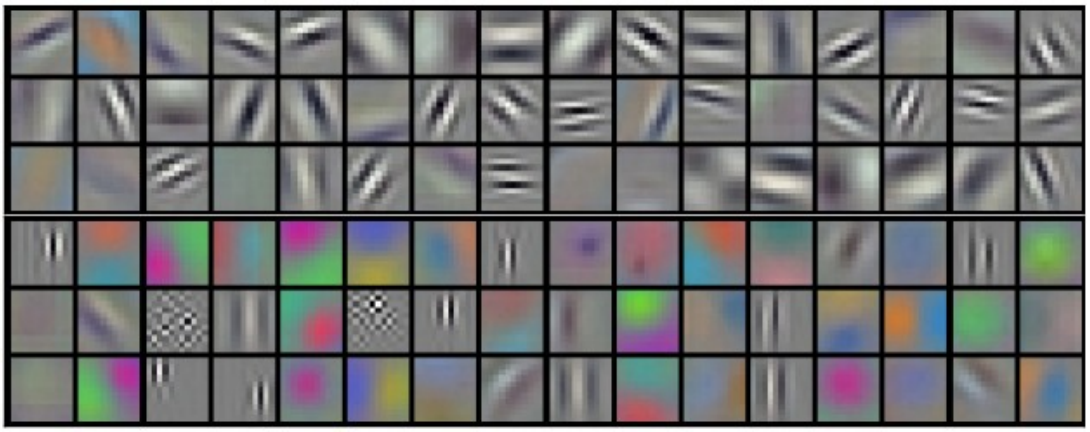

Figure 2.5: Filters taken from [Alex Krizhevsky & Hinton 12]. It is visible that different learned filters focus on different features including frequencies, edges at different orientations, phases, and colors. Best viewed in color.

Convolutional Layer

Convolutional layer is the core building block of a CovNet. The parameters of the CovNet are learn-able filters or kernels. Every filter is small spatially but extends through the full depth of the input volume. During a forward pass (see Section 2.2.2) the filter is convolved across the width and the height of the input volume computing the *dot* product at each location. This produces 2-dimensional activation maps representing the responses of the filters at every spatial location. For an input volume I of size $N \times N \times c$ and filter F of size $m \times m$, the convolution operation and the convolution operator $*$ are defined as follows,

$$I * F = C(x, y) = \sum_{i=1}^{c}(\sum_{y_k=0}^{N} (\sum_{x_k=0}^{N} I(x - x_k, y - y_k, i)F_i(x_k, y_k)))$$

where C is the output of the convolution operation.

The network will thus learn filters that activate on encountering some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

Natural images have the property of being stationary i.e, statistics of one part of the image is similar to some other part. This suggests that features learned from one part of the image can be applied to another part. *Parameter sharing* in convolutional layers is based on this property and it is used to control the number

of parameters and thus making CovNets translationally invariant. This is implemented by constraining the neurons at each depth slice to use same weights and bias.

Three hyper-parameters control the size of the output volume of the convolutional layer.

- **Depth** of the output volume controls the number of neurons in the layer that connect to the same region of the input volume. It corresponds to the number of filters to be learned in the layer.
- **Stride** tells how many pixels should be skipped while sliding the filter during the convolution operation.
- **Zero-padding** directly allows to control the size of the output volume.

The convolution operation is an integral transform and it is therefore also a linear operator that transforms data from one domain to another. This allows a CovNet to learn to transform data using visual features into domains where separation can be achieved by a hyperplane.

Pooling Layer

Pooling layer is a form of non-linear down-sampling. *Max-pooling* is the most common type of pooling layer used in modern CoveNets. It partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum. The pooling layer reduces the computational burden by reducing the number of parameters hence addressing *over-fitting* and it also introduces scale invariance in the network. Pooling layers are commonly placed between successive convolutional layers. An example of the max pooling is shown in Figure. 2.6.

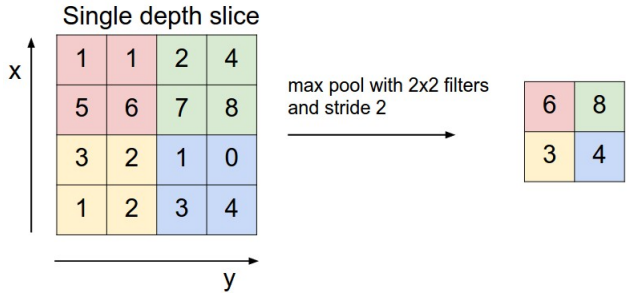

Figure 2.6: The most common pooling operation is max-pooling shown with a filter size of 2×2 and a stride of 2. Thus each max is taken over 4 numbers which are shown as colored boxes.

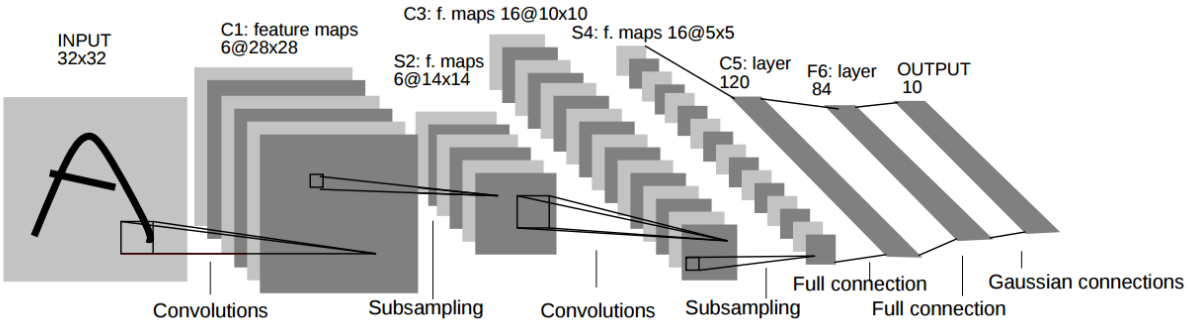

Figure 2.7: A convolutional neural network architecture for the task of handwritten text recognition. The network shows convolutional layers, sub-sampling layers (pooling) and fully connected layers stacked. Taken from [LeCun & Bottou$^+$ 98].

Fully Connected Layer

Neurons in a fully connected layer have full connections to all the activations in the previous layer. They are responsible for high-level reasoning and are thus usually connected to the end of CovNets.

Loss Layer

The loss layer drives the overall learning of the CovNet. The loss layer has a loss function which has the goal of learning by mapping parameter configuration to a scalar value that specifies the fitness of the parameters. Hence, the goal of learning is to find a configuration of the weights that minimizes the loss function.

2.2.2 Training CovNets

Training convolutional neural networks consists of two steps: forward passing the input and backward passing the gradient.

Forward Pass (Forward Propagation)

During the forward pass of the convolutional layer each filter is convolved across the width and height of the input volume producing the 2-D activation maps. Let the input volume z^l from the previous layer be of size $N \times N \times c$ and the filter F be of size $m \times m$ with depth d, then the output size is given by $(N-m+1) \times (N-m+1) \times d$.

The convolution operation is applied as follows for each convolutional layer,

$$z_{x,y}^{l+1} = F^{l+1} * \phi(z_{x,y}^{l}) + bias_{x,y}^{l+1}$$

where $*$ is the convolution operation defined in Section 2.2.1 and ϕ is an activation function (see Section 2.1.1). The activation function is applied to all the d maps to add some non-linearity and to produce the input volume for the next layer.

The convolutional layer is mostly followed by a max pooling layer (see Section 2.2.1). During the forward pass of the max pooling layer, the dimensionality of the input is reduced depending on the kernel size. For a kernel size 2 and a stride size 2, an input of size $N \times N$ would be reduced to $N/2 \times N/2$.

Fully connected layers are often added at the end of convolutional layers for high level reasoning. Let w_{jk}^{l} denote the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. Additionally, let the bias be $bias_{j}^{l}$ of the j^{th} neuron in the l^{th} layer and let the activation a_{j}^{l} of the j^{th} neuron in the l^{th} layer. Then the forward pass in the fully connected layers is applied as follows,

$$z_{j}^{l+1} = \sum_{k} w_{jk}^{l+1} a_{k}^{l} + bias_{j}^{l+1}$$

where $a^{l} = \phi(z^{l})$ and ϕ is the activation function.

Backward Pass (Backward Propagation)

Backward propagation [LeCun Yan & Denker 89] is a layer wise application of chain rule for partial gradients. The purpose of back propagation is to determine the best parameters which yields the minimum error of the objective function.

The error at convolutional layer l is defined using the partial derivatives as,

$$\delta_{(x,y)}^{l} = \frac{\partial J}{\partial z_{x,y}^{l}} = \sum_{x'} \sum_{y'} \frac{\partial J}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^{l}}$$

where J is the objective function or the cost function.

By replacing the first term by the definition of error and expanding $z_{x',y'}^{l+1}$ we can write,

$$\delta_{(x,y)}^{l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial(\sum_{a} \sum_{b} F_{a,b}^{l+1} \phi(z_{x'-a,y'-b}^{l}) + bias_{x',y'}^{l+1})}{\partial z_{x,y}^{l}}$$

Since the *bias* does not depend on $z^l_{x,y}$, we can also write the same expression as,

$$\delta^l_{(x,y)} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} F^{l+1}_{a,b} \phi^{'}(z^l_{x,y})$$

Using $a = x' - x$ and $b = y' - y$, the equation can be reformulated to,

$$\delta^l_{(x,y)} = \sum_{x'} \sum_{y'} \delta^{l+1}_{x',y'} F^{l+1}_{x'-x,y'-y} \phi^{'}(z^l_{x,y})$$

Using the definition of convolution (see Section 2.2.1), the final equation becomes,

$$\delta^l_{(x,y)} = \delta^{l+1}_{x,y} * F^{l+1}_{-x,-y} \phi^{'}(z^l_{x,y})$$

Finally, the gradient with respect to the filters is computed for updating the filters,

$$\frac{\partial J}{\partial F^l_{a,b}} = \sum_{x} \sum_{y} \frac{\partial J}{\partial z^l_{x,y}} \frac{\partial z^l_{x,y}}{\partial F^l_{a,b}}$$

Using the same reasoning as before, it can be shown that the equation reduces to,

$$\frac{\partial J}{\partial F^l_{a,b}} = \delta^l_{a,b} * \phi^{'}(z^{l-1}_{-a,-b})$$

These equations can be used to update the filters using any of the optimization techniques described in Section 2.1.4.

Backward propagation in pooling layers is rather simple. Since there is no learning involved in the pooling layers themselves, no parameters need to be updated. In forward propagation each $k \times k$ block is reduced to single value. This single value acquires an error computed from backwards propagation from the previous layer. This error is then just forwarded to the place where it came from. Since it only came from one place in the $k \times k$ block, the back propagated errors from max-pooling layers are rather sparse.

For understanding the back propagation algorithm in fully connected layers, we first define the *Hadamard product* $\bigodot$. For two vectors s and t, the Hadamard product is defined as,

$$(s \bigodot t)_j = s_j t_j$$

The error δ^l_j of neuron j in layer l in a fully connected layer is defined as,

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}$$

where J is the objective function or the cost function and z_j^l is defined in the previous section. The error at the output layer l can be calculated as,

$$\delta^l = \nabla_a J \bigodot \phi^{'}(z^l)$$

where $\nabla_a J$ represents the rate of change of J with respect to the output activations. Using this equation we can represent the error δ^l in terms of the error in the next layer δ^{l+1},

$$\delta^L = ((w^{l+1})^T \delta^{l+1}) \bigodot \phi(z^l)$$

where $(w^{l+1})^T$ is the transpose of the weight matrix w^{l+1} for the $(l+1)^{th}$ layer. The equation for the rate of change of the objective function to any bias in the network is given by,

$$\frac{\partial J}{\partial bias_j^l} = \delta_j^l$$

And the equation for the rate of change of the objective function with respect to any weight in the network is given by,

$$\frac{\partial J}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

These equations can be used with any of the optimization strategy described in previous sections to update the weights. For further reading on back propagation in fully connected layers we refer the reader to [LeCun 92].

2.2.3 Training Techniques In CovNets

Batch Normalization In CovNets

Batch normalization [Ioffe & Szegedy 15] (see Section 2.1.5) can also be applied to convolutional layers. Let F be the filters to be learned, I^l be the input to the layer from the previous layer, b be the bias and ϕ be the activation function used, then the operation performed by the layer is given by,

$$z = \phi(F * I^l + b)$$

The batch normalization transform can be added just before the non-linearity by normalizing $x = F * I^l + b$. The bias term can be removed from this expression since it will be canceled during the normalization step. Thus the final operation of the convolution layer becomes,

$$z = \phi(BN(F * I^l))$$

where BN refers to the batch normalization transform as described in Section 2.1.5.

Data Augmentation

Plentiful high-quality data is the key to learning good models. Smart approaches to data augmentation can increase the size of the training while still keeping the quality of the data set. Using data augmentation can often make the model more robust, prevent overfitting and generalize to unseen environments.

There are many approaches to augment data. Some of the simple approaches which can be used are random rotations, random translations, random cropping, zoom, flips and color perturbations. Some of the advanced schemes can include creating synthetic data. [Rogez & Schmid 16] suggested a two stage process for creating synthetic data for 3D pose estimation. The first step was a MoCap-guided mosaic construction stage that stitches image patches together and the second step was the pose aware blending stage that improves image quality and erases patch seams. They show that networks trained on this unrealistic data will still generalize very well to existing datasets.

2.3 Convolutional Pose Machines

Pose machine [Varun Ramakrishna & Sheikh 14] is an inference framework for articulated human pose estimation. The pixel location of the p-th body joint is referred to as a part, $Y_p \in \mathrm{Z} \subset \mathbb{R}^2$, where Z is the set of all (u, v) locations in the image. Pose machine consists of a sequence of multi-class predictors that are trained to predict the location of each part in each level of the hierarchy. In each stage $t \in \{1..T\}$, the classifiers g_t predicts beliefs for assigning a location to each part based on the features extracted from the image at a certain location and contextual information from the preceding classifier in the neighborhood around the location in stage t. The pose machine uses random forests for prediction ($\{g_t\}$), fixed hand-crafted image features across all stages and fixed hand crafted context feature maps to capture spatial context across all stages.

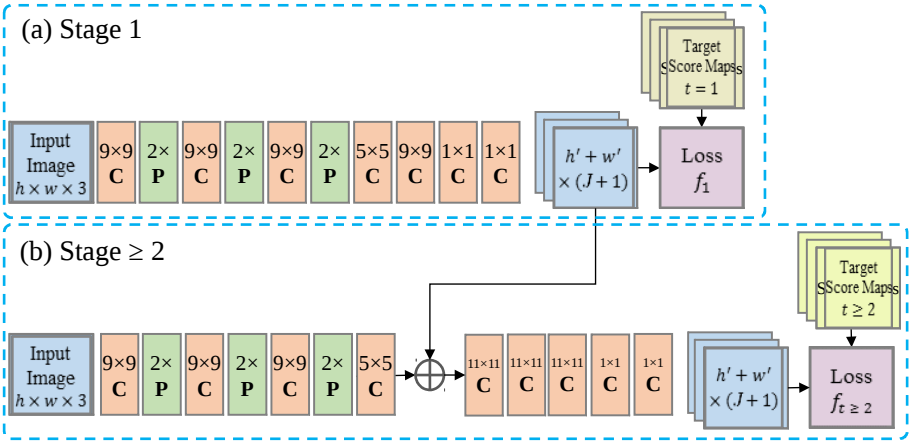

Figure 2.8: CPM architecture taken from [Shih-En Wei & Sheikh 16]. The first stage network structure is composed of five convolutional layers followed by two 1x1 convolutional layers resulting in a fully convolutional architecture. The heat maps generated from the first stage are fused with the output of the first part of the stage ≥ 2. The estimates of the joint locations are increasingly refined at each stage. All stages are locally supervised and a separate loss is computed for each stage.

Convolutional pose machines (CPM) [Shih-En Wei & Sheikh 16] show that the inference model in pose machines [Varun Ramakrishna & Sheikh 14] can be replaced by a deep convolutional architecture. This allows for both image and contextual information to be learned directly from the data. Another advantage of using convolutional architecture is that the architecture is completely differentiable and thus allows for an end to end training of all stages of the convolutional pose machine.

2.3.1 Architecture

The architecture of the CPM is shown in Figure 2.8. Given an input image I of size $w \times h$ containing the subject, the 2D pose of the subject is defined as a set $\chi = \{x_j\}_{j=1...J}$ for $J = 14$ body joints where $x_j \in \chi$ represents the 2D location (u, v) of the j^{th} joint in the image. Each stage in CPM is a multi label classifier $\phi_t(x)$ that is trained to provide confidence scores $s_t^j \in \mathbb{R}^{w \times h}$ for each joint $j \in J$. The first stage of the CPM consists of small receptive fields. The part detection at this stage is done by using only the local evidence i.e, the network is constrained to a small patch around the output pixel location providing the confidence scores,

$$\phi_{t=1}(\mathrm{x}|I) \rightarrow \{s_1^j(\mathrm{x}_j = \mathrm{x})\}_{j=1..J+1}$$

The receptive field at the output layer of the second stage network is large enough to allow the learning of potentially complex and long range correlations between the joint locations. Large receptive fields can either be achieved by using the pooling layer at the expense of precision or by increasing the kernel size at the expense of drastically increasing the network parameters. CPM uses multiple convolutional layers to achieve a large receptive field on the 8x downscaled heat maps from the previous stage. In addition to the local image evidence, all subsequent stages also utilize the contextual information from the preceding stages to produce confidence score maps,

$$\phi_{t>1}(\mathrm{x}|\mathrm{I}, \psi(\mathrm{x}, \mathrm{s}_{\mathrm{t}-1})) \rightarrow \{s_t^j(\mathrm{x}_j = \mathrm{x})\}_{j=1..J+1}$$

where $\psi(\mathrm{x}, s_{t-1})$ indicates the mapping from the scores s_{t-1} to the context features for location x.

To achieve the best precision, the size of the input images is scaled to a square size of 368×368.

2.3.2 Tackling The Problem Of Vanishing Gradients

Training a deep network with many layers is prone to the problem of vanishing gradients. As it was observed by [Bradley 10] and [Glorot & Bengio 10], when the number of layers increase in a network, the magnitude of the back propagated gradient decreases in strength. This means that neurons in the earlier layers learn much more slowly than the neurons in the later layers. CPM addresses this problem by repeatedly producing the belief maps for the location of each of the parts. This is enforced by defining a loss function at the output of each stage t that minimizes the l_2 distance between the predicted and ground truth belief map for each part. This intermediate supervision has the advantage that even though the full architecture can have many layers, it does not fall prey to the vanishing gradient problem as the intermediate loss functions replenish the gradients at each stage.

2.3.3 Learning In Convolutional Pose Machines

The ground truth belief map of a part p is defined as $b_*^p(Y_p = z)$. These are generated by putting Gaussian peaks at ground truth location of each skeleton part. The objective function to be minimized is therefore defined as follows,

$$f_t = \sum_{p=1}^{P+1} \sum_{z \in Z} ||b_t^p(z) - b_*^p(z)||_2^2$$

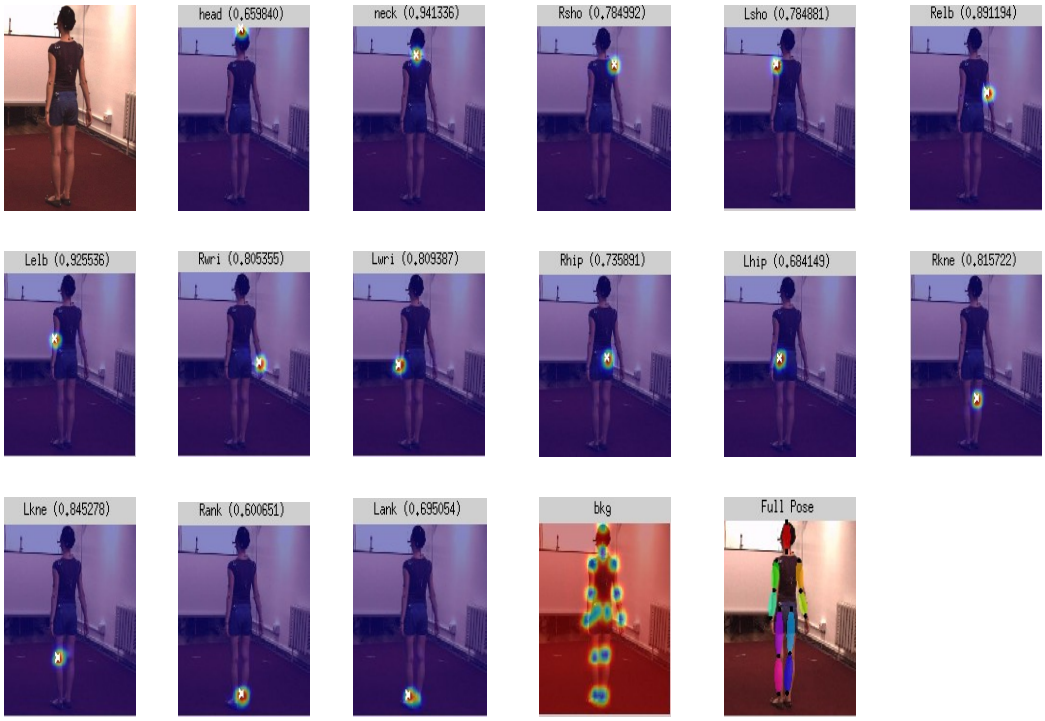

Figure 2.9: The figure shows an example run of the convolutional pose machine for $t = 2$ stages which already gives quite good results for 2D pose estimation. The first image is the original image which is fed to the CPM. The output produced are the belief maps for each joint part $p \in J$.

Since there is a loss layer after the end of each stage, the overall objective function is obtained by summing up the individual losses at each stage. It is given by,

$$\mathcal{F} = \sum_{t=1}^{T} f_t$$

Stochastic gradient descent (SGD) (see Section 2.1.4) was the optimization technique used in CPM to jointly train all the T stages in the network.

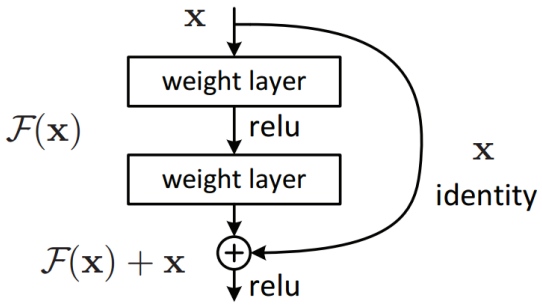

Figure 2.10: A building block of a residual network from [He & Zhang$^+$ 15]

2.4 Residual Networks

Vanishing gradients is a difficulty found in training deep neural networks with gradient based methods. In particular, the earlier layers in the network are the worst affected. Since the gradient back propagated to the earlier layers is extremely small, these layers learn very slowly. This means that even a large change in the value of the parameters in the earlier layers causes a very small change in the network's output. This problem is very evident in training significantly deep networks since it hampers their convergence. The depth of representation is of central importance for many vision tasks but due to the problem of vanishing gradients training very deep networks was previously unachievable. [Andrew M. Saxe & Ganguli 14] [Glorot & Bengio 10] addressed this issue by normalizing the initial layers and [Ioffe & Szegedy 15] (see Section 2.1.5) addressed the problem by normalizing the intermediate layers. As noted by [He & Zhang$^+$ 15], however even when the deeper networks are able to start converging they face the degradation problem: when the network depth increases, accuracy gets saturated and decreases rapidly.

Residual networks [He & Zhang$^+$ 15] present a residual learning framework which allows and eases the training of substantially deep networks. It also addresses the problem of degradation in deep networks by explicitly letting the layers learn the residual mapping instead of directly fitting the desired underlying mapping.

2.4.1 Architecture

Let $\mathcal{H}(\mathrm{x})$ be the underlying mapping to be learned by a few stacked layers where x denotes the input to the first layer of these stacked layers. Rather than letting the stacked layers to approximate $\mathcal{H}(\mathrm{x})$, the residual networks let the layers explicitly approximate the residual function $\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$ and thus the original function then becomes $\mathcal{F}(\mathrm{x}) + \mathrm{x}$. [He & Zhang$^+$ 15] argue that its easier to optimize this

residual mapping instead of the original unreferenced mapping.

In feed-forward networks $\mathcal{F}(\mathrm{x}) + \mathrm{x}$ can be formulated using *skip* connections or *shortcut* connections. These skip connections perform an identity mapping and their output is added to the output of the stacked layers. A building block of a residual network with a single skip connection is shown in Figure 2.10. The skip connections do not introduce any extra parameters or additional computational complexity.

The skip connections can be added after every few stacked layers to construct a very deep network which is somewhat immune to the problem of the vanishing gradients and the degradation problem. An example 34-layer architecture is shown in Figure 2.11 which achieved state of the art results on the ImageNet classification challenge [Russakovsky & Deng$^+$ 14]. The 152-layer residual network constructed in similar manner won the first prize in the $ILSVRC$ 2015 classification competition.

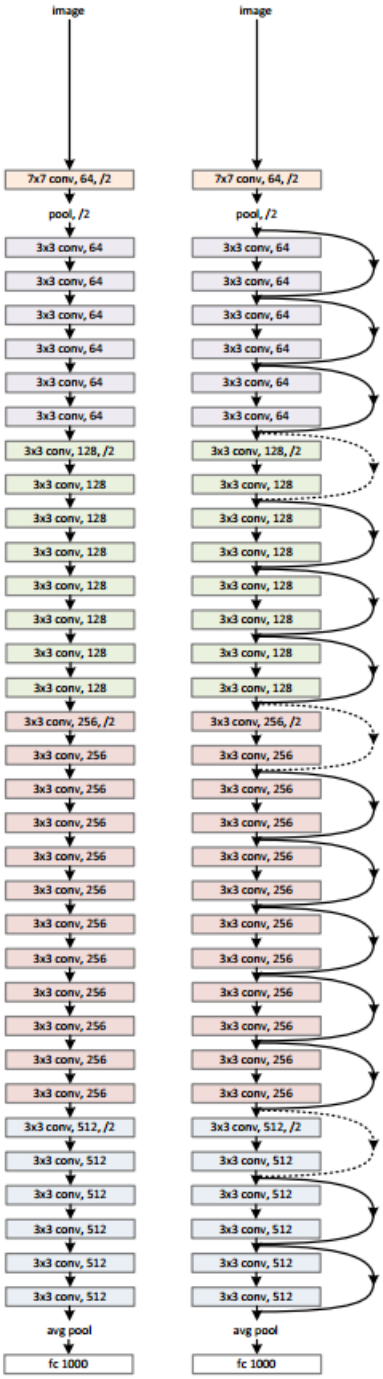

Figure 2.11: Left: A simple 34-layer network without any skip connections. Right: A 34-layer Residual network with multiple skip connections which achieved state of the art results on the ImageNet challenge from [He & Zhang$^+$ 15]

3 Methods and Approach

In this section we describe the methods and approaches applied in the course of the thesis. Firstly the motivation behind the each approach is described and then the detailed explanation of each experiment follow. The hardware used for each experiment is also mentioned. We start by introducing the reader to Human3.6M dataset. A large scale publically available dataset we used for evaluating our approach.

3.1 Human3.6M Dataset

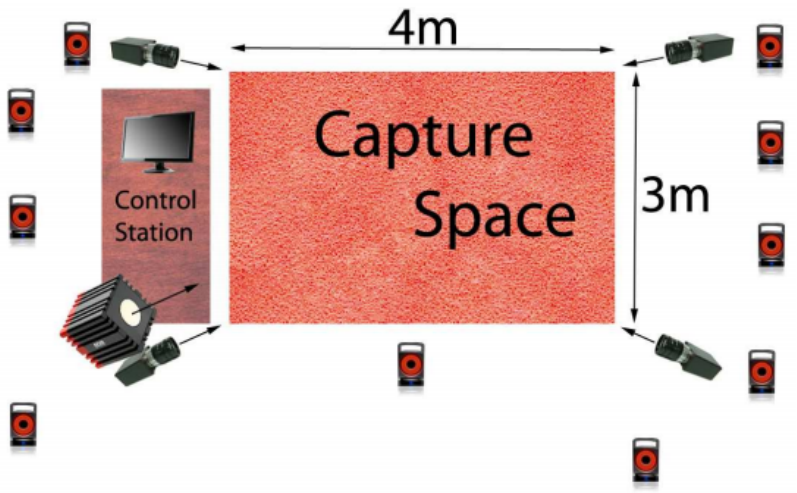

Figure 3.1: The designated laboratory area used was 6m× 5m of which 4m×3m was the effective capture space where all the actors were fully visible in all the cameras. The figure also shows the placement of the video, MoCap and Time of flight cameras. From [Catalin Ionescu & Sminchisescu 14].

Human 3.6M [Catalin Ionescu & Sminchisescu 14] is a large scale dataset containing accurate 3D human poses recorded under a synchronized setup. The dataset records the performance of 11 professional actors: 5 female and 6 male. The body mass index (BMI) of the actors spanned from 17 to 29 providing a great amount of body

shape variability. The data was captured using 15 sensors. Four digital cameras were used to capture the data from 4 different angles, one time of flight camera for depth images and ten motion capture cameras were used. The laboratory and hardware setup is shown in Figure 3.1.

Since the human kinematic space is too large to be sampled regularly and densely, the dataset focuses on collecting data from the common urban and office scenes. The pose data is collected from 15 scenarios where actors were allowed to wear their normal clothing and were given general instructions about the scenarios giving them freedom to improvise. The 15 scenarios performed by the actors are directions, discussion, eating, sitting, sitting Down, greeting, taking photo, posing, making purchases, smoking, waiting, walking, sitting on chair, talking on the phone, walking the dog and walking together.

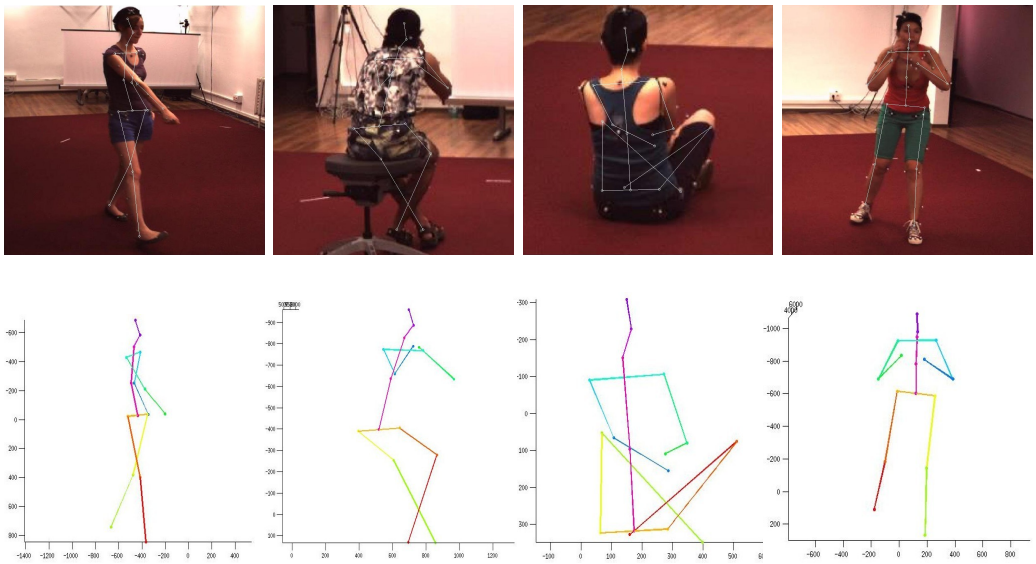

Figure 3.2: The figure shows some of the example actions being performed by 4 different actors from the Human3.6m dataset. The first row shows the images with 2D poses plotted on the images and the second row shows the corresponding 3D poses of the actions.

Figure 3.2 shows some example data provided in the dataset. Other than the images, 3D poses, 2D poses and TOF data, the dataset also provides software for background subtraction, extracting the bounding boxes of the actors, methods for visualization of the skeleton based on the 3D joint locations as well as 3D joint angles. A variety of pre-computed features are also provided as part of the dataset

to allow rapid experimentation and evaluation.

3.2 Evaluation Protocol

Human 3.6m dataset provides actions from 11 different subjects $S_{1..11}$ performing complex motion scenarios based on typical human activities. Following the standard protocol of [Bugra Tekin & Fua 16], [Sungheon Park & Kwak 16], [Li & Chan 14] and [Xiaowei Zhou & Daniilidis 16] the data from 5 subjects $(S_1, S_5, S_6, S_7, S_8)$ was used for training in all the experiments and the data from 2 different subjects (S_9, S_{11}) was used for testing.

3.3 Implementation Details

In the experiments, the aim was to directly regress the 3D human pose from a single RGB image. The pose was represented in terms of the 3D locations $y \in \mathbb{R}^{J\times 3}$ of J body joints relative to the root joint which was the torso. The number of joints used were 17. An alternative could have been to use angle between the limbs and limb lenghts to represent the pose but this is a less homogenous representation and it is rarely used. The implementation was done using Caffe deep learning framework [Jia & Shelhamer$^+$ 14]. The setup of parameters and training protocol of each of the approaches are described below and the comparison with the state of the art results follow in the next sections.

3.4 Training for Each Activity

Following the protocol of [Li & Chan 14], [Bugra Tekin & Fua 16] and [Sungheon Park & Kwak 16], we trained a smaller network specifically for each activity present in the Human3.6M dataset. The architecture of our three convolution network is shown in Figure 3.3. The individual filter sizes of the convolutional layers are 9×9 each with a 2×2 pooling layer to follow after each convolutional layer. This network is the first 3 convolutional layers of the first stage of the convolutional pose machine (see Section 2.3). After the convolutional layers we have two fully connected layers of 4096 parameters each and then the 51 parameter fully connected layer follow which gives us the final pose prediction of 17×3 coordinates. We select 4096 as the number of parameters for our fully connected layers after taking inspiration from DeepCut [Pishchulin & Insafutdinov$^+$ 16]. We used the weights of the first three convolution layers of the first stage of the convolutional pose machine as our initialization and an euclidean loss to fine tune the network. The activation function

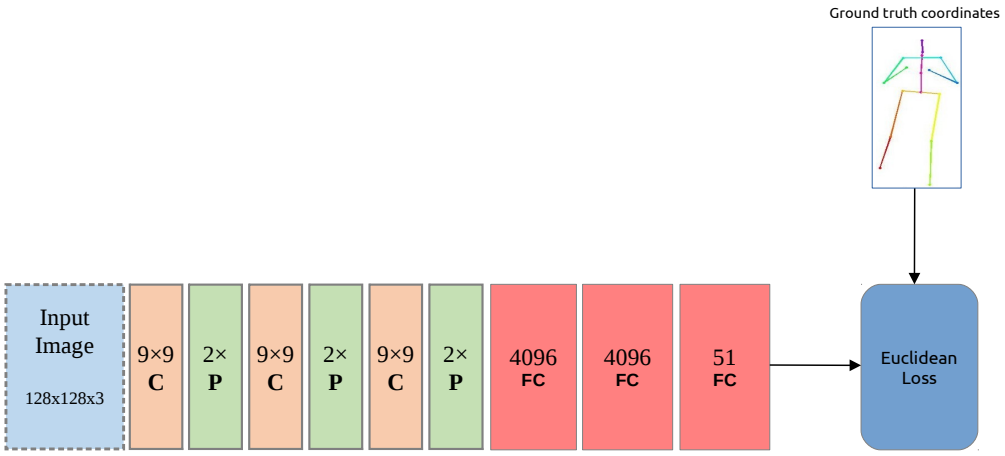

Figure 3.3: Our network architecture for training for each activity.

used was the ReLu function (see Section 2.1.1). To reduce overfitting dropout layer was also used between each fully connected layer with the dropout probability of 0.5.

The method for calculating the bounding box provided as part of the dataset was used to crop a subject centered image from the original image. The cropped image was then resized to 150×150. Following the protocol of [Bugra Tekin & Fua 16] and [Sungheon Park & Kwak 16] random cropping of size 128×128 from the subject centered image were used as input to the network. The input image was mean substracted before feeding to the network. The target ground truth 3D poses were also first mean subtracted and then normalized such that the $frobenius$ norm of each 3D pose was 1. Since there are some variations in the scale of different actors in the dataset, normalizing the coordinates helps to reduce the ambiguity of scale and predict scale invariant poses. This is the same protocol used by [Sungheon Park & Kwak 16] and a similar protocol is used by [Xingyi Zhou & Wei 16]. The base learning rate was set to 0.0001 and it was reduced to half after every 4 *epochs*. The momentum was set to 0.9 and the weight decay was set to 0.005. These hyperparameters were carefully selected after manual search using different experimentation and finding the best values for the task at hand. We used the *Adam* (see Section 2.1.4) optimizer as our optimization strategy. The batch size set during training was 128. The training was complete after 28 *epochs* and took around 10-12 hours depending on the size of the training data available for each activity. The hardware used during training was an Intel Core(TM) i7-5930K CPU @ 3.50GHz with 64 GB RAM and a nvidia Titan X GPU.

During the test time, it is essential to recover the original scale of the pose to evaluate the error measure and be able to compare with the state of the art results. In our case the original scale was inferred from the training data. Specifically, the scale for each actor was calculated by averaging the length of all connected joints in the training data and dividing by the average length of all connected joints in the test data for that actor. Since the lengths of the limbs often have a large variation, only the lengths of the joints in the torso which are more stable were used to recover the original scale. The torso joints used to recover the original scale of the actor during test time are shown in Figure 3.4. Since we randomly cropped an image of size 128×128 from a 150×150 sized image during training, during testing the subject centered image was also resized to 150×150 but only a center crop of size 128×128 was taken for evaluating the error metric.

Figure 3.4: The Figure highlights the torso joints used in recovering the scale of the actor at test time. The joints in red are only used since they have much less variation than the limb joints. Best viewed in color.

We refer to this approach as $Direct\text{-}regression_{each\ activity}$ in our evaluation section.

3.5 Training for All Activities

Following the protocol of [Xingyi Zhou & Wei 16] we did several experiments to train a single network for all the activities present in the Human3.6M dataset. Specifically, we investigate the effect of using 2D pose information either on the fly in form of 2D part heat maps or by adding an additional loss layer for the task of 2D pose estimation. We explain each of the approaches in detail in the following sections.

3.5.1 Single Stage CPM Network

We first train a single seven convolution baseline architecture for the task of 3D human pose estimation on all the activities present in the Human3.6M dataset.

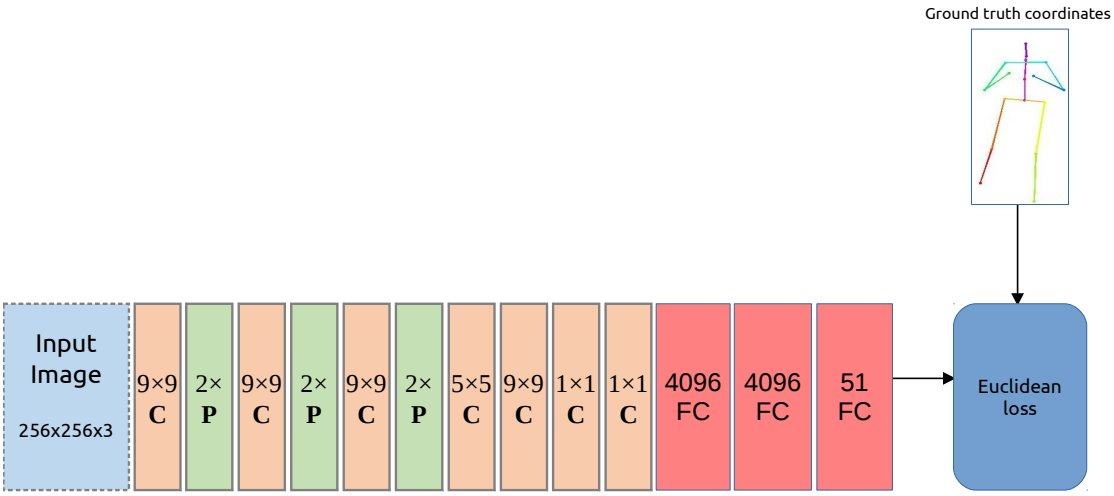

Figure 3.5: Our single stage CPM network without the '2D prediction stage'. This network served as a check whether 2D heat maps add any robustness and increase accuracy for the task of 3D pose estimation or not.

The complete network architecture is shown in Figure 3.5. This architecture is a single stage of the convolutional pose machine [Shih-En Wei & Sheikh 16].

The total training data available for the 5 subjects is about 1.5 million images. They are highly similar and redundant. Thus, following the protocol of [Xingyi Zhou & Wei 16], we selected $800k$ random images from all activities for training. No data augmentation was used in this case. The bounding boxes given in the dataset were used to extract a square region with the subject centered and the resulting image was resized to 256×256. A larger image size than the previous experiment was used for training the network because as claimed by [Shih-En Wei & Sheikh 16], CPM works best at 368×368 image size for predicting the 2D poses, we tested the reduced sizes with CPM and selected 256×256 at the end as they gave very reasonable 2D pose estimates and allowed us to train faster on this reduced size images. The input images were then mean subtracted before feeding them to the network. The target ground truth 3D coordinates were also mean subtracted and normalized such that the $frobenius$ norm of the coordinates was 1 similar to our $Direct\text{-}regression_{each\ activity}$ experiment.

The same hardware was used to train this network as the previous experiment. The base learning rate was set to 0.0001 which was reduced to half after every *epoch*. The momentum was set to 0.9 and the weight decay was set to 0.0005. As in the previous experiment, the hyper parameters were tuned after a manual

search. The activation function used was the ReLu function (see Section 2.1.1) and a dropout layer (see Section 2.1.3) was used between each fully connected layer with the dropout probability of 0.5. We used the *Adam* optimizer as out optimization strategy. We trained the network for 4 *epochs*. The training took less than 3 days to complete.

During the testing phase, the original scale of the 3D poses for each actor was recovered in a similar manner as described for the $Direct\text{-}regression_{each\ activity}$ experiment. The subject centered image was resized to 256×256 but this time no cropping were done. The fourth channel is then added to the testing image which contains a Gaussian blob at object position using the same *sigma* of 27 as in training.

We will refer to this approach as $Single\text{-}Stage^{cpm}_{all\ activities}$ in our evaluation section.

3.5.2 Two Stage CPM Network

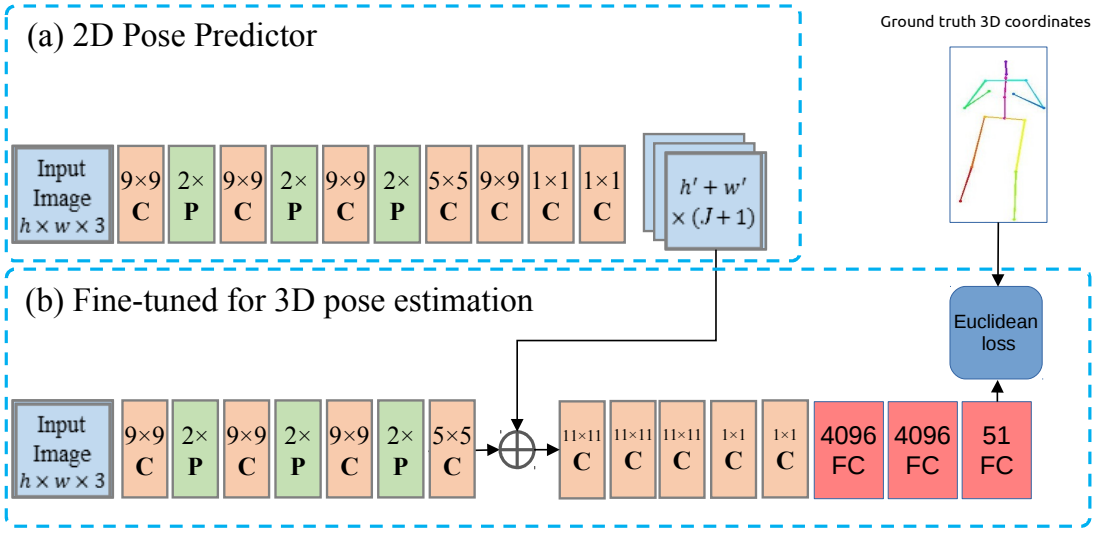

Figure 3.6: Two stage CPM network with a single euclidean loss for 3D poss estimation. The first stage provides 2D pose maps which are then combined with the second stage to jointly extract features for 3D pose estimation.

In this experiment we investigate the effect of using 2D pose information to regress the 3D pose. In our approach we use a two stage network taken from the convolutional pose machine framework [Shih-En Wei & Sheikh 16]. The first stage of the network produces 2D probability maps or heat maps for each joint. The predicted heat maps are then fused in the second stage of the network which is finetuned

for the task of 3D pose estimation. The goal of our experiment is to increase the robustness and accuracy of 3D pose estimation from a single image by exploiting 3D image cues to the full while making use of the 2D joint locations predicted by the previous stage.

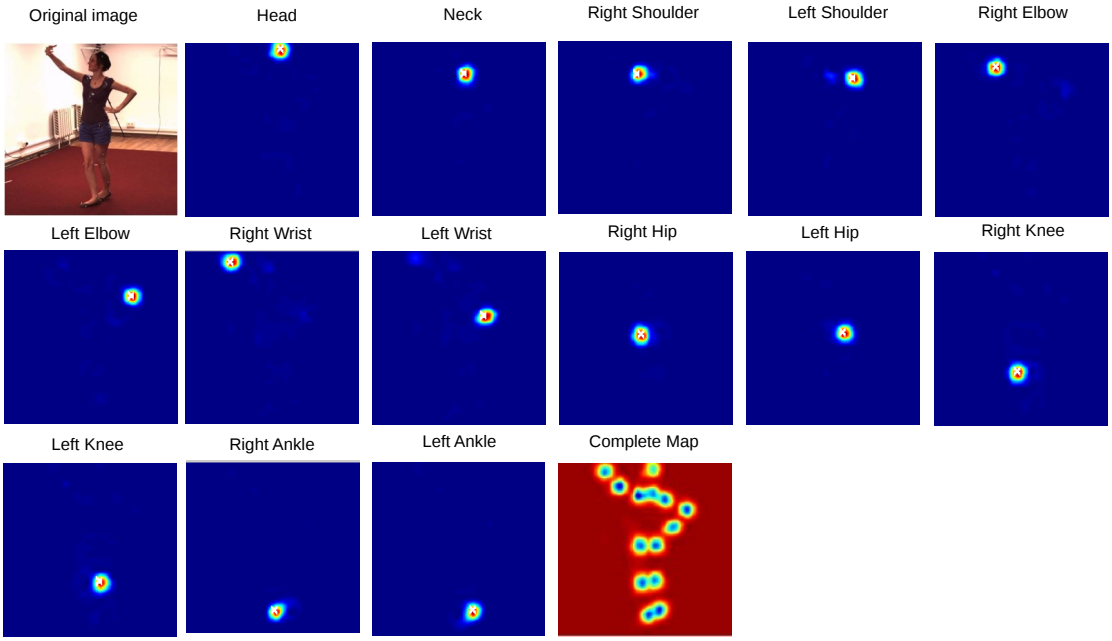

Figure 3.7: The heat maps generated by the first stage of the two stage CPM network which are combined with the second stage of the network to further learn features for 3D pose estimation.

The complete architecture of the two stage CPM network is shown in Figure 3.6. The first stage of the network purely works as a 2D pose predictor providing the 2D pose information to the second stage which is fine tuned for the task of 3D pose estimation. For this purpose we use the weights of the original trained CPM from [Shih-En Wei & Sheikh 16] and we set the learning rate of the first stage to 0. The first stage produces $J+1$ heat maps containing the 2D pose information which is then combined with the second stage and further features are extracted by the following five convolutional layers as shown in Figure 3.6. After the convolutional layers we have two fully connected layers each of 4096 parameters followed by the 51 parameter fully connected layer which gives us the final prediction of 17×3 3D pose coordinates. We used the euclidean loss to fine tune the second stage of the network for the task of 3D pose estimation. We believe that 2D pose information

is a very strong prior to predict the 3D coordinates and thus this information could help to better generalize the trained network.

We set the network parameters same as in the $Single\text{-}Stage^{cpm}_{all\ activities}$ experiment. For speed, we train this network on two nvidia Titan X GPUs with an effective batch size of 80. The network is trained for a total of 8 *epochs* and the base learning rate is reduced to half after each *epoch*. During the testing phase the same protocol is used as before for recovering the original scale of the actor. For preparing the test image, we first crop a subject centered image and resize it to the size of 256×256. A fourth channel is then added to the input image. This fourth channel contained a Gaussian blob at the center of the subject location. This is a pre-requisite of using the 2D pose estimator of the CPM since it is done in the original CPM paper. The *sigma* value of 27 was used to generate the Gaussian blob as in the original work. The image is also mean subtracted before feeding to the network.

We refer to this approach as $Two\text{-}Stage^{cpm}_{all\ activities}$ in our evaluation section.

3.5.3 Two Stage CPM Network With Multiple Losses

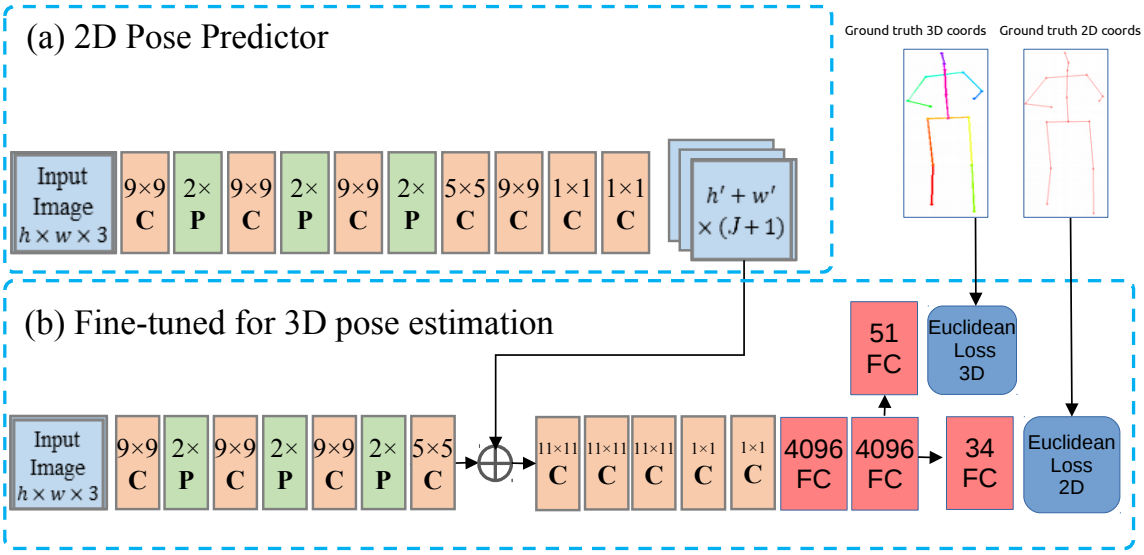

Figure 3.8: Two stage CPM network with an additional loss for 2D pose estimation. The second stage of the network learns joint features for both 2D and 3D pose estimation and thus increasing the complexity of the task. We hope to regularize the 3D pose estimation using the 2D pose loss and help avoid over fitting in the network.

Taking inspiration from [Sungheon Park & Kwak 16], we train our two stage CPM network with both 2D loss and 3D loss. We investigate whether jointly learning features for 2D and 3D can help us better generalize the network. The complete architecture of the network is shown in Figure 3.8.

The hyper parameters of the network were set to the same values as before in the two stage CPM network (see Section 3.5.2). Additionally the loss weight of the 3D euclidean loss λ_{3D} was set to 0.8 and the loss weight λ_{2D} of the 2D euclidean loss was set to 0.2 for the first 4 *epochs*. The loss weight λ_{2D} of the 2D euclidean loss was then changed to 0.01 and the training was further done for 4 more *epochs*. The final loss in the network is calculated as a weighted sum of the individual losses. Let L_f be the final loss calculated during training, L_i be the *i-th* loss produced in the network and let λ_i be the weight associated with the *i-th* loss. Then the final loss of the network is calculated as follows,

$$L_f = \sum_{i=1}^{n} \lambda_i L_i$$

where n is the number of losses produced by the network. The higher weight of a certain loss in the network forces the network to 'focus' more on that loss and a stronger gradient is back propagated from that loss. Our primary task is the task of 3D human pose estimation thus we set the weight for its corresponding loss higher.

During the testing phase we use the same protocol as in the previous experiment $Two\text{-}Stage^{cpm}_{all\ activities}$ for preparing the testing images and recovering the original scale of the actor.

We refer to this approach as $Two\text{-}Stage\text{-}Multiloss^{cpm}_{all\ activities}$ in our evaluation section.

3.5.4 Resnet 50-Layer Architecture For 3D Pose Estimation

Residual networks [He & Zhang$^+$ 15] (see Section 2.4) has shown that deep neural networks can be trained successfully. Deep neural networks have more representational power and this power is gained from hierarchically composing shallower feature representations into deeper representations. For instance, in face recognition, pixels make edges and edges make corners. Corners define facial features such as eyes, noses, mouth and chin and facial features compose to define faces. [He & Zhang$^+$ 15] showed the power of residual networks by winning the 2015 ImageNet challenge [Russakovsky & Deng$^+$ 14].

Residual networks have also been used to train end to end 3D human pose estimation models. [Xingyi Zhou & Wei 16] used a 50-layer residual network that was

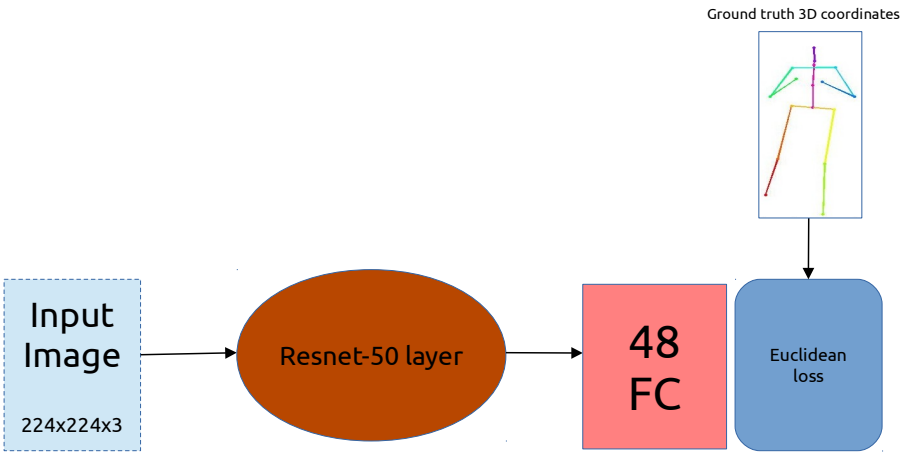

Figure 3.9: Resnet 50 layer architecture for 3D human pose estimation. The resnet was pre-trained on ImageNet [Russakovsky & Deng$^+$ 14].

pre-trained on ImageNet [Russakovsky & Deng$^+$ 14]. This 50-layer network was fine-tuned on the task of 3D human pose estimation. The architecture of the model is shown in figure 3.9. One thing to note is that the network was trained to output $J = 16$ joints rather than $J = 17$ where the torso joint is missing. [Xingyi Zhou & Wei 16] explained that there are historic reasons to use 16 joints for regression with different versions of kinematic human models. The 17-*th* joint is thus calculated from neck and pelvis joint using the ratio defined by the bone lengths.

[Xingyi Zhou & Wei 16] trained a single model for all the activities present in the Human 3.6m dataset. A total of $800k$ images were selected at random from all the activities and no data augmentation was used. A square region with the subject in the center was cropped from the original image using the bounding box information provided in the dataset and the resulting image was resized to 224×224. The images were mean subtracted before feeding them to the network. The ground truth 3D pose coordinates were also normalized using the bounding box size i.e, by dividing by the longer side of the bounding box. The network was trained for 70 *epochs* with the base learning rate of 0.003 which was dropped to 0.0003 after 50 *epochs*. A batch size of 52 was used on 2 GPUs and the momentum was set to 0.9 and the weight decay was set to 0.0002. [Xingyi Zhou & Wei 16] referred to this approach as their baseline network for 3D pose estimation and we will refer to it as [Xingyi Zhou & Wei 16]$_{baseline}$ in our comparisons.

Another contribution of [Xingyi Zhou & Wei 16] is to introduce a kinematic layer to serve as a geometrically more accurate loss layer which they used in their final

configuraion to achieve state of the art results on human 3D pose estimation. Let the rotation angle of the i-th joint be θ, the motion parameter Θ includes the global position $\mathbf{p}$, global orientation $\mathbf{o}$, and all the rotation angles, $\Theta = \{\mathbf{p}, \mathbf{o}\} \cup \{\theta_i\}_{i=1}^{J}$. The forward kinematic function is a mapping from motion space to joint location space,

$$\mathcal{F} : \{\Theta\} \rightarrow \mathcal{Y}$$

where $\mathcal{Y}$ is the coordinate for all joints $\mathcal{Y} \in \mathcal{R}^{3\times J}$. The new loss function is defined as,

$$L(\Theta) = \frac{1}{2}||\mathcal{F}(\Theta) - Y||^2$$

where $Y \in \mathcal{Y}$ is the ground truth joint locations in the input image. We will refer to this approach of theirs as [Xingyi Zhou & Wei 16]$_{final}$ in our comparisons.

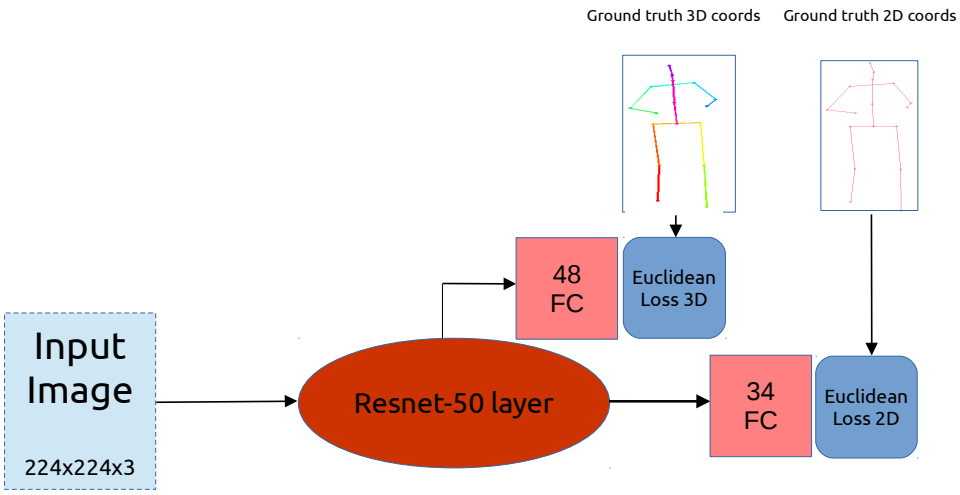

Figure 3.10: Our 50-layer Resnet architecture with 2D and 3D loss. We investigate the effect of jointly learning the features for both 2D and 3D pose estimation and check whether this increases the robustness and accuracy of the network.

Inspired by the work of [Xingyi Zhou & Wei 16], we fine-tuned the 50-layer residual network for the task of both 2D and 3D pose estimation to investigate whether the features learned jointly for 2D and 3D pose estimation help to better generalize the model and give better results for 3D pose estimation or not. For this purpose, we

used the same setup as used by [Xingyi Zhou & Wei 16] as described before and further added a loss for 2D pose estimation. The network architecture is shown in Figure 3.10. The loss weight for the 3D pose euclidean loss was set to 0.5 and the loss weight for the 2D pose euclidean loss was set to 0.1. The total training was done for 10 *epochs*. The base learning rate was set to 0.003 which was reduced to 0.0003 after 5 epochs. The loss weight of the 2D pose euclidean loss was also reduced to 0.01 after 5 *epochs* so as to favor the learning for 3D pose estimation. The weight decay and the momentum were set to the same value as before. We used the stochastic gradient descent (SGD) (see Section 2.1.4) optimization technique as our optimizer. The hardware used for training was the same as before and we used two nvidia Titan X GPUs. The batch size was set to 33 on a single GPU thus our effective batch size was 66.

During the testing phase, we use a similar protocol as before. The testing image was cropped with the subject in the center and resized to 224×224. Since the network produces the output of J =16 joints instead of J =17 joints, we calculated the $17th$ joint from the neck and pelvis joint using the ratio defined by the bone lengths in the training set. This is done to make the comparison fair with the other experiments since all the other experiments output J=17 joints.

We refer to this approach as $Resnet\text{-}Multiloss_{all\ activities}$ in our evaluation section.

3.5.5 Resnet 50-Layer Architecture For 2D Pose Estimation

For completion, we also trained a 50-layer residual network for only the task of 2D pose estimation. The network architecture is shown in Figure 3.11. With this experiment we investigate if the network trained for learning joint features for both 2D and 3D pose under-performs or out-performs the network trained for only 2D pose estimation. We train the network in the similar environment as the last experiment with multiple losses. We use $800k$ randomly selected images from the training set of almost 1.5 million images. We similarly as before crop the subject centered image and resize it to 224×224 size. The images are mean subtracted before feeding them to the network. The target 2D poses are also mean subtracted and normalized by using the bounding box size i.e, they are divided by the size of the bounding box. The network is trained on the same hardware as before using a single nvidia Titan X GPU. The batch size is set to 34. The training was done for a total of 10 *epochs*. We used the trained weights of the network by [Xingyi Zhou & Wei 16] as our initialization for a fair comparison. The base learning rate was set to 0.003 for the first 5 *epochs* and it was changed to 0.0003 for the next 5 *epochs*. The momentum was set to 0.9 and the weight decay was set to 0.0002. We used the stochastic gradient descent (SGD) optimization technique as our optimizer.

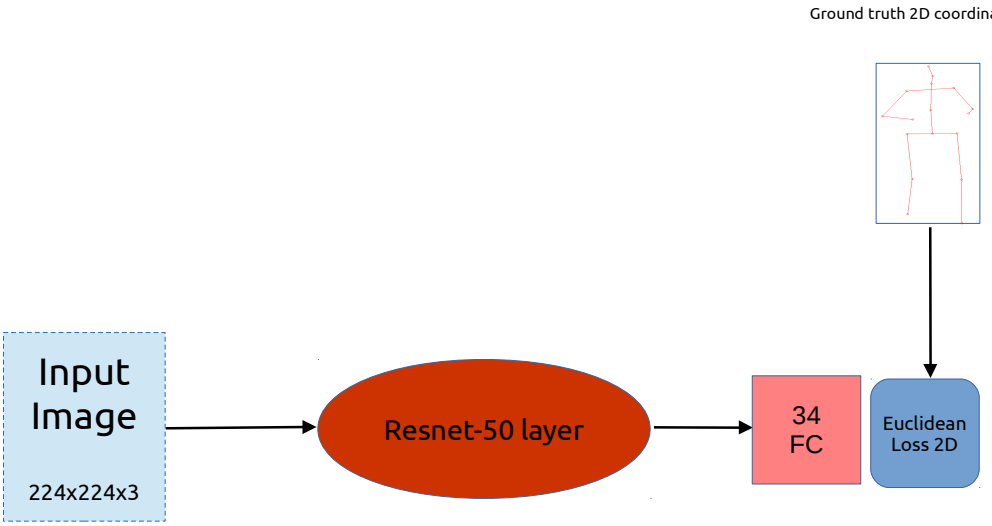

Figure 3.11: Our Resnet 50 layer with 2D loss. We use the weights of network provided by [Xingyi Zhou & Wei 16] as the initialization.

During the testing phase, the same protocol is used as before. The subject centered image is cropped and resized to 224×224 size. The network produces 17×2 2D pose estimates. We refer to this approach as $Resnet^{2D}_{all\ activities}$ in our evaluation section.

3.5.6 Summary of Methods and Approaches

Experiment	Pre-trained on	Num. of Convs	3D loss used	2D loss used	Trained for
$Direct\text{-}regression_{each\ activity}$	[Shih-En Wei & Sheikh 16]	3	yes	no	Each activity
$Single\text{-}Stage^{cpm}_{all\ activities}$	[Shih-En Wei & Sheikh 16]	7	yes	no	All activities
$Two\text{-}Stage\text{-}Multiloss^{cpm}_{all\ activities}$	[Shih-En Wei & Sheikh 16]	16	yes	no	All activities
$Two\text{-}Stage\text{-}Multiloss^{cpm}_{all\ activities}$	[Shih-En Wei & Sheikh 16]	16	yes	yes	All activities
$Resnet\text{-}Multiloss_{all\ activities}$	[Xingyi Zhou & Wei 16]	50	yes	yes	All activities
$Resnet^{2D}_{all\ activities}$	[Xingyi Zhou & Wei 16]	50	yes	yes	All activities

Table 3.1: The table shows the summary of the experiments done during the course of the thesis and described in the previous sections.

4 Evaluation and Discussion

4.1 Error Metrics

Mean per joint position error (MPJPE) was used as the error metric to evaluate the networks following the protocol of [Bugra Tekin & Fua 16], [Sungheon Park & Kwak 16], [Li & Chan 14], [Xiaowei Zhou & Daniilidis 16] and [Catalin Ionescu & Sminchisescu 14] to make a fair comparison. Let $m^{(f)}_{\mathrm{f},\mathcal{S}}(i)$ be a function that returns the 3D coordinates of the *i-th* joint of skeleton $\mathcal{S}$, at frame f and from the pose estimator f. Let also $m^{(f)}_{gt,\mathcal{S}}$ be the *i-th* joint of the ground truth frame f. Then the MPJPE error metric is defined as,

$$E_{MPJPE}(f,\mathcal{S}) = \frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} ||m^{(f)}_{\mathrm{f},\mathcal{S}}(i) - m^{(f)}_{gt,\mathcal{S}}(i)||_2$$

where $N_{\mathcal{S}}$ is the number of joints in skeleton $\mathcal{S}$. For a set of frames the error is the average over the individual errors of all the frames.

Some of our experiments described in the previous sections also produce 2D poses as output together with the 3D poses and in some cases 2D poses alone. We use the average squared difference in pixels between the ground truth and predicted pixel location of the joints as our evaluation metric. The metric is formally defined as,

$$E_{SQDIFF}(\mathcal{S},i) = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathcal{F}}} ||m^{(f)}_{\mathrm{f},\mathcal{S}}(i) - m^{(f)}_{gt,\mathcal{S}}(i)||_2$$

where $m^{(f)}_{\mathrm{f},\mathcal{S}}(i)$ and $m^{(f)}_{gt,\mathcal{S}}$ are the same quantities as described before but for 2D poses rather than 3D poses. And $N_{\mathcal{F}}$ is the total number of frames in the testing set. One thing to notice here is that the error is not averaged over the whole skeleton $\mathcal{S}$ but it is averaged over each joint over the whole testing set.

4.2 Evaluation of Training on Each Activity

Figure. 4.1 shows the training error obtained by applying our approach of $Direct\text{-}regression_{each\ activity}$ on each activity in the Human3.6M dataset. We trained each

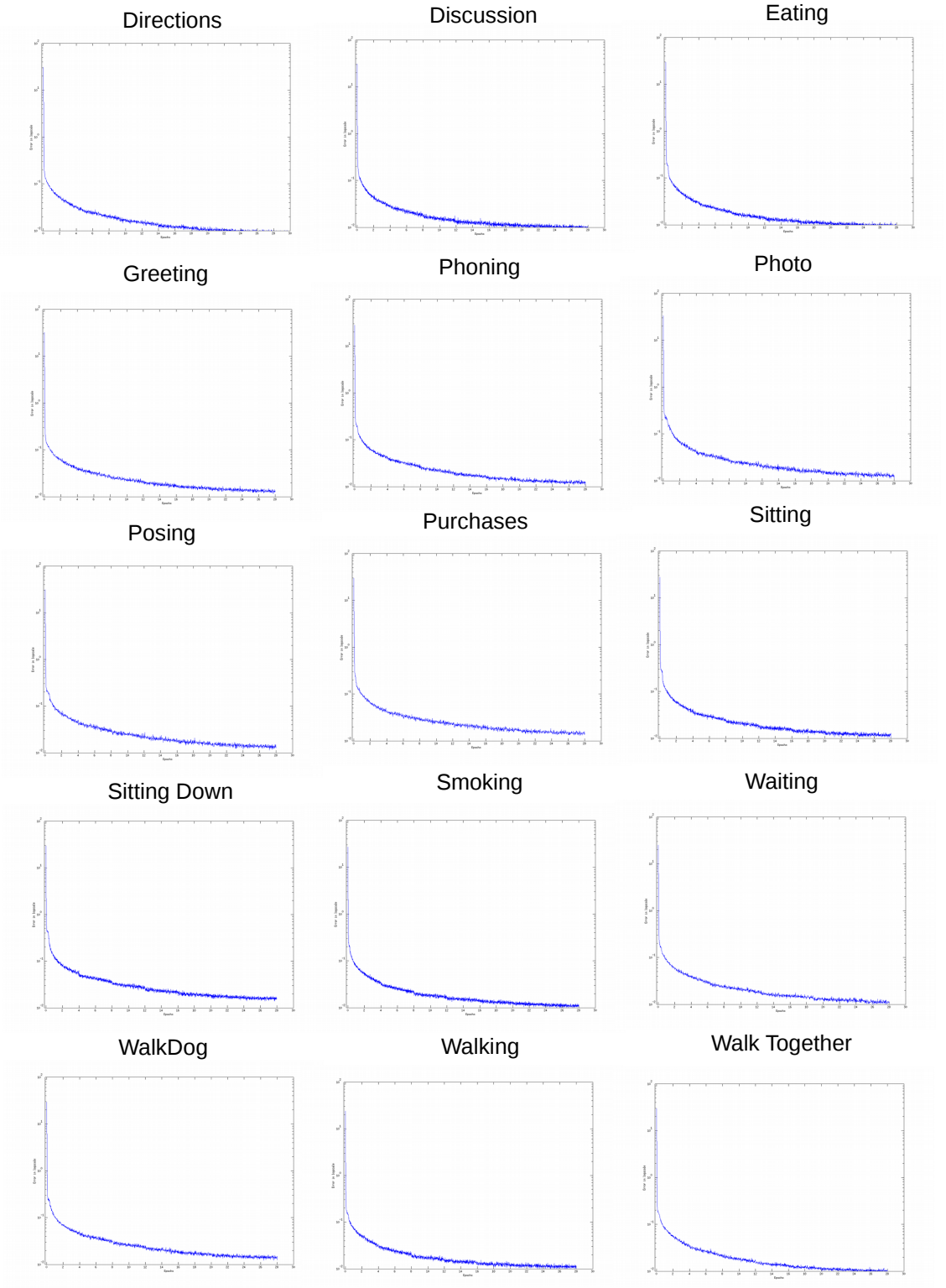

Figure 4.1: The Figure shows the training error for all the activities in the Human3.6M dataset using our $Direct\text{-}regression_{each\ activity}$ approach.

	Directions	Discussion	Eating	Greeting	Phoning	Photo
[Li & Chan 14]	-	148.79	103.31	127.17	-	190.37
[Bugra Tekin & Fua 16]	-	129.06	91.43	121.68	-	162.17
[Tekin & Rozantsev^{+} 15]	102.41	147.72	88.83	125.28	118.02	182.73
[Sungheon Park & Kwak 16]	100.34	**116.19**	89.96	116.49	**115.34**	149.55
Direct-regression$_{each\ activity}$(Ours)	**91.86**	120.34	**86.98**	**112.38**	121.62	**145.76**
	Posing	Purchases	Sitting	SittingDown	Smoking	Waiting
[Li & Chan 14]	-	-	-	-	-	-
[Bugra Tekin & Fua 16]	-	-	-	-	-	-
[Tekin & Rozantsev^{+} 15]	112.38	129.17	138.89	224.90	118.42	138.75
[Sungheon Park & Kwak 16]	117.57	**106.94**	**137.21**	**190.82**	**105.78**	**125.12**
Direct-regression$_{each\ activity}$(Ours)	**101.77**	111.92	137.86	210.02	113.85	130.68
	WalkDog	Walking	WalkPair	Average		
[Li & Chan 14]	149.59	77.60	-	-		
[Bugra Tekin & Fua 16]	130.53	65.75	-	-		
[Tekin & Rozantsev^{+} 15]	126.29	**55.07**	**65.76**	124.97		
[Sungheon Park & Kwak 16]	131.90	62.64	96.18	**117.34**		
Direct-regression$_{each\ activity}$(Ours)	**120.37**	70.12	95.27	118.05		

Table 4.1: Results of Human3.6M dataset by using the protocol of training a separate network for each activity. The error metric is the E_{MPJPE} (see Section 4.1) which is the mean euclidean distance in mm between the ground truth 3D joints and the estimations of different methods.

of the networks separately for each of the activities as described before. For a fair evaluation, the results are only compared with the papers which follow the same protocol of training a separate network for each activity. The results are shown in Table 4.1. We see that we achieve state of the art results using our *Direct-regression*$_{each\ activity}$ approach. The total average in mm for our approach was 118.05, only second to [Sungheon Park & Kwak 16] which was 117.34. Looking at the table it is visible that we achieve better results on some of the activities than the other approaches. We attribute these results to the good tuning of the hyper parameters, excellent weight initialization provided by [Shih-En Wei & Sheikh 16] and appropriate selection of the number of parameters for the fully connected layers in our approach. Some of the qualitative results produced by our approach of *Direct-regression*$_{each\ activity}$ are shown in Figure 4.2.

4.2.1 Two Stage CPM Network

Since the described network has a huge number of parameters, we trained one network for all the activities.

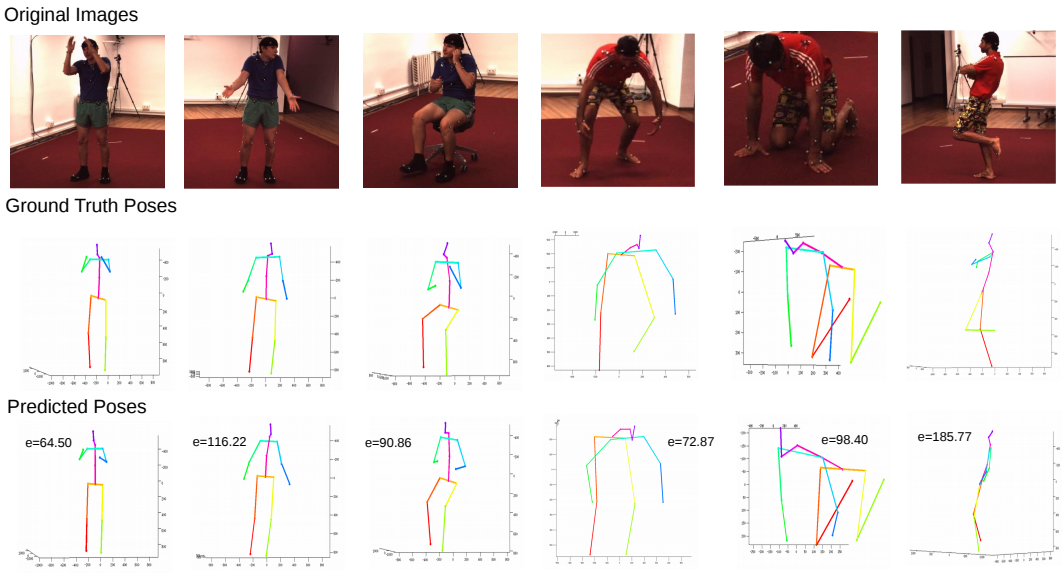

Figure 4.2: The Figure shows the qualitative results of our approach of *Direct-regression*$_{each\ activity}$. The first row shows the original image, the second row shows the ground truth poses and the third row shows the predicted poses with the error in *mm*.

The network was evaluated for each of the 15 activities present in the Human3.6M dataset. The results are reported in table 4.2. The results are reported in *mm* using the E_{MPJPE} error metric as described in section 4.1.

4.2.2 Single Stage CPM Network

To investigate whether 2D pose information indeed helps in predicting the 3D pose robustly and accurately and better generalize our two stage CPM network, we also trained a single stage CPM network without the *2D prediction part*. The network was trained in exactly the same setting as the two stage CPM network. The hyper parameters were set to the same value and the network was trained for 4 *epochs*. We trained the network on a single GPU rather than two GPUs since we could already fit a batch size of 70 on 1 GPU. The complete training took a little less than 3 days to complete.

During the testing phase, the same protocol was used as in the two stage CPM network. The input image was resized to 256×256 and then the fourth channel was added with the Gaussian blob at the subject location. The original scale of the actors were retrieved in the same manner as before and the network was evaluated for all the 15 activities present in the Human3.6M dataset. The results in *mm* are reported in the table 4.2 using the error metric E_{MPJPE}.

4.2.3 Two Stage CPM Network With Multiple Losses

In our two stage CPM network with a single euclidean loss for 3D pose estimation we saw that the network started to over fit after 4 *epochs* of training. To tackle the problem of over fitting in our network we add a second loss layer for 2D pose estimation to jointly train the whole network for both 3D and 2D pose estimation. This was done to add a sort of regularization for the 3D pose estimation and to increase the complexity of the task so that over fitting could be avoided. The network architecture is shown in figure 3.8. We trained a single network for all the 15 activities present in the Human3.6M dataset.

During testing the same protocol is used as in the two stage CPM network with single euclidean loss. The subject centered test image is resized to 256×256 and then the fourth channel is added with the Gaussian blob at the location of the subject with *sigma* value of 27. The network is evaluated for each activity and the results are reported in mm in the table 4.2 using the error metric E_{MPJPE}.

We also evaluate the 2D pose coordinates predicted by the network. The network produces 17×2 2D pose coordinates. We use the error metric E_{SQDIFF} as described in section 4.1. The results for the 2D pose evaluation are reported in table 4.3.

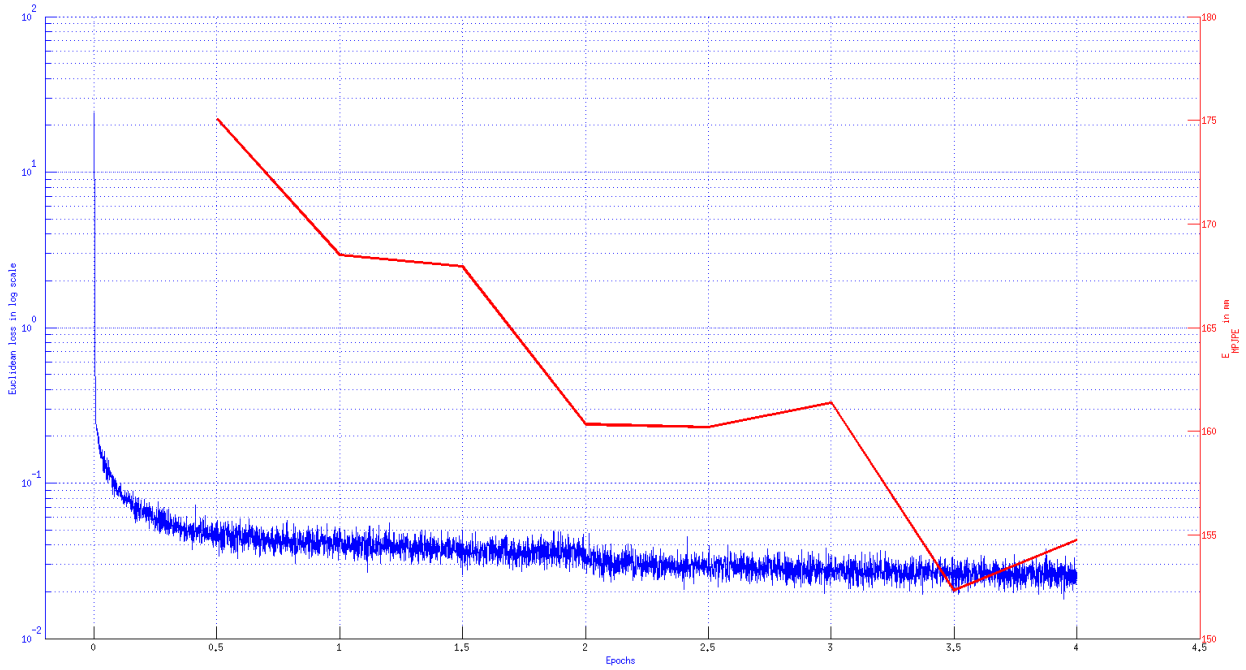

Figure 4.3: salam

4.2.4 Resnet 50-Layer Architecture For 3D Pose Estimation

We also evaluate the 2D poses predicted by the network. The network produces 17×2 2D pose coordinates. We use the error metric E_{SQDIFF} as described in section 4.1. The results for the 2D pose evaluation are reported in table 4.3.

4.2.5 Resnet 50-Layer Architecture For 2D Pose Estimation

We use the error metric E_{SQDIFF} as described in section 4.1. The results for the 2D pose evaluation are reported in table 4.3.

4.3 Results

In this section we describe the results of our experiments. We first give a detailed comparison of our results with the state of the art 3D human pose results and then

in the following section we do a qualitative analysis of our different experiments.

4.3.1 Comparison With State Of The Art

	Directions	Discussion	Eating	Greeting	Phoning	Photo
[Xingyi Zhou & Wei 16]	91.83	102.41	96.95	98.75	113.35	125.22
[Xingyi Zhou & Wei 16] baseline	106.38	104.68	104.28	107.80	115.44	114.05
Ours 3_conv	115.77	136.17	106.62	112.38	137.77	151.68
Ours CPM full	99.93	126.83	110.66	106.29	119.33	138.79
Ours CPM 7 conv	138.17	155.78	157.53	147.64	171.02	176.23
Ours CPM with 2d loss	126.66	147.79	143.81	131.04	141.66	168.12
Ours Resnet with 2d loss	168.53	186.33	142.79	158.56	155.53	170.66
	Posing	Purchases	Sitting	SittingDown	Smoking	Waiting
[Xingyi Zhou & Wei 16]	90.04	93.84	132.16	158.97	106.91	94.41
[Xingyi Zhou & Wei 16] baseline	103.80	109.03	125.87	149.15	112.64	105.37
Ours 3_conv	108.28	162.99	158.30	210.78	126.44	130.81
Ours CPM full	100.59	137.42	151.51	217.15	116.38	107.87
Ours CPM 7 conv	146.82	171.58	205.31	233.62	150.69	154.38
Ours CPM with 2d loss	141.41	154.96	166.84	204.80	137.13	132.28
Ours Resnet with 2d loss	158.87	181.30	201.27	171.99	161.52	161.84
	WalkDog	Walking	WalkPair	Average		
[Xingyi Zhou & Wei 16]	126.04	79.02	98.96	107.26		
[Xingyi Zhou & Wei 16] baseline	113.69	98.19	110.17	112.03		
Ours 3_conv	141.37	95.08	102.75	133.15		
Ours CPM full	130.18	92.82	103.94	123.98		
Ours CPM 7 conv	159.37	137.52	165.88	164.77		
Ours CPM with 2d loss	143.62	113.21	143.62	146.46		
Ours Resnet with 2d loss	177.82	148.97	157.85	166.92		

Table 4.2: Results of Human3.6M dataset. The error metric is the E_{MPJPE} (see section 4.1) which is the mean euclidean distance in mm between the ground truth 3D joints and the estimations of different methods.

Table 4.2 shows the evaluation of different academic works in comparison to our experiments. Specifically, we compare our results with [Bugra Tekin & Fua 16], [Sungheon Park & Kwak 16] and [Xingyi Zhou & Wei 16]. An overview of their approaches can be found in the Related work section (see 1.2). For the work of [Xingyi Zhou & Wei 16], we consider two of their experiments in our comparison. The first one is their baseline experiment of directly regressing the 3D poses using a 50-layer residual network which is pre-trained on the ImageNet. The second experiment we consider from them is their final architecture with a post processing kinematic layer after the 50-layer residual network with which they achieve the best

average results. The results from all the comparison methods are taken from their published papers.

From the table 4.2 it is visible that we achieve the best results using the two stage CPM network with a single euclidean loss for 3D pose estimation. This two stage network out performs the single stage CPM network which is without the '2D prediction' stage on all the activities present in the Human3.6M dataset. This shows us that 2D pose heat maps from the first stage of the network architecture did help to make the network more robust and accurate. One thing to notice here is that both of these networks were trained under exactly the same environments and were trained for the same number of *epochs*.

Our second best results come from our baseline three convolution network. It has to be noticed that there was a separate network trained for each of the activity and then it was evaluated for the testing data of the same activity thus these networks have the edge that they are better adapted for predicting the poses for the specific activity. The three convolutional baseline networks have a disadvantage that each network has to be trained separately which takes around 7-10 hours for each activity depending on the amount of training data available thus it will approximately take around a whole week to train each network sequentially. Our two stage CPM network with a single euclidean for 3D pose estimation just requires around 2.5 days to train and a single network can predict poses for each activity.

We tested our two stage CPM network with multiple losses both for 2D pose estimation and 3D pose estimation. We train this network in the same environment as the two stage CPM network with a single euclidean loss and for the same amount of *epochs*. We see that this network fails to perform good and only beats our best performing network on one activity thus showing that the network was not able to learn very meaningful joint features for both 2D and 3D pose estimation.

Our 50-layer residual network with both 2D and 3D loss also fails to perform very well and does not beat our best performing architecture on any activity. Since we fine tuned the network of [Xingyi Zhou & Wei 16] with an additional loss for 2D pose estimation, the network was already very well tuned for the task of 3D pose estimation. Adding another loss forced the network to learn joint features for both 2D and 3D pose estimation and the network was not able to generalize very well for the joint task.

The over all best performing method is the work of [Xingyi Zhou & Wei 16]. The disadvantage of their network is that it is very time consuming to train. They train the network for 70 epochs using a batch size of 52 on 2 GPUs. According to our estimation such a network would take around a whole week to train. Our best performing network takes around 2.5 days to train and still achieve comparable results on each activity.

	Head	Chin	Abdomen	Torso	Neck	Left Wrist
Ours Resnet multi loss						
Ours Resnet only 2D						
Ours CPM multi loss						
	Left Elbow	Left Shoulder	Right Shoulder	Right Elbow	Right Wrist	Left Ankle
Ours Resnet multi loss						
Ours Resnet only 2D						
Ours CPM multi loss						
	Left Knee	Left Hip	Right Hip	Right knee	Right Ankle	Average
Ours Resnet multi loss						
Ours Resnet only 2D						
Ours CPM multi loss						

Table 4.3: Results of the 2D pose estimation evaluated on the Human3.6M dataset. The numbers are mean squared euclidean distance in pixels between the ground truth joints and the estimations of different methods.(numbers will be added soon)

4.3.2 Qualitative Analysis

In this section we compare the results of our networks qualitatively with the ground truth poses.

The Task Of 3D Pose Estimation

Figure 4.4 shows each of our experiment evaluated on five different images of the test subjects S_9 and S_{11}. The error E_{MPJPE} for each of the evaluated poses is also shown in the figure. We see that on average the error of our two stage CPM network with a single euclidean loss is the lowest. As described in the previous section, the second lowest error on average is of the three convolution baseline network. Since each of this network was specifically trained for each activity, this makes sense. Our other three networks namely, the single stage CPM network, the two stage CPM network with multiple losses and the 50-layer residual network with multiple losses perform almost the same with sometimes one network beating the other on some image.

The Task Of 2D Pose Estimation

The qualitative results for the 2D pose estimation by some of our different Resnet architectures is shown in figure 4.5. From the table 4.4 we see that the 50-layer residual network performs quite well when only trained for the task of 3D pose estimation and does not perform too well when it is jointly trained for both 3D

and 2D pose estimation tasks. A similar trend can also be seen in figure 4.5. Our 50-layer residual network architecture does not perform too well either on the 2D poses when trained jointly for 2D and 3D pose estimation but the 2D poses are quite accurate when the network is only fine tuned for the task of 2D pose estimation. The empirical results of both of the networks are shown in the table 4.3.

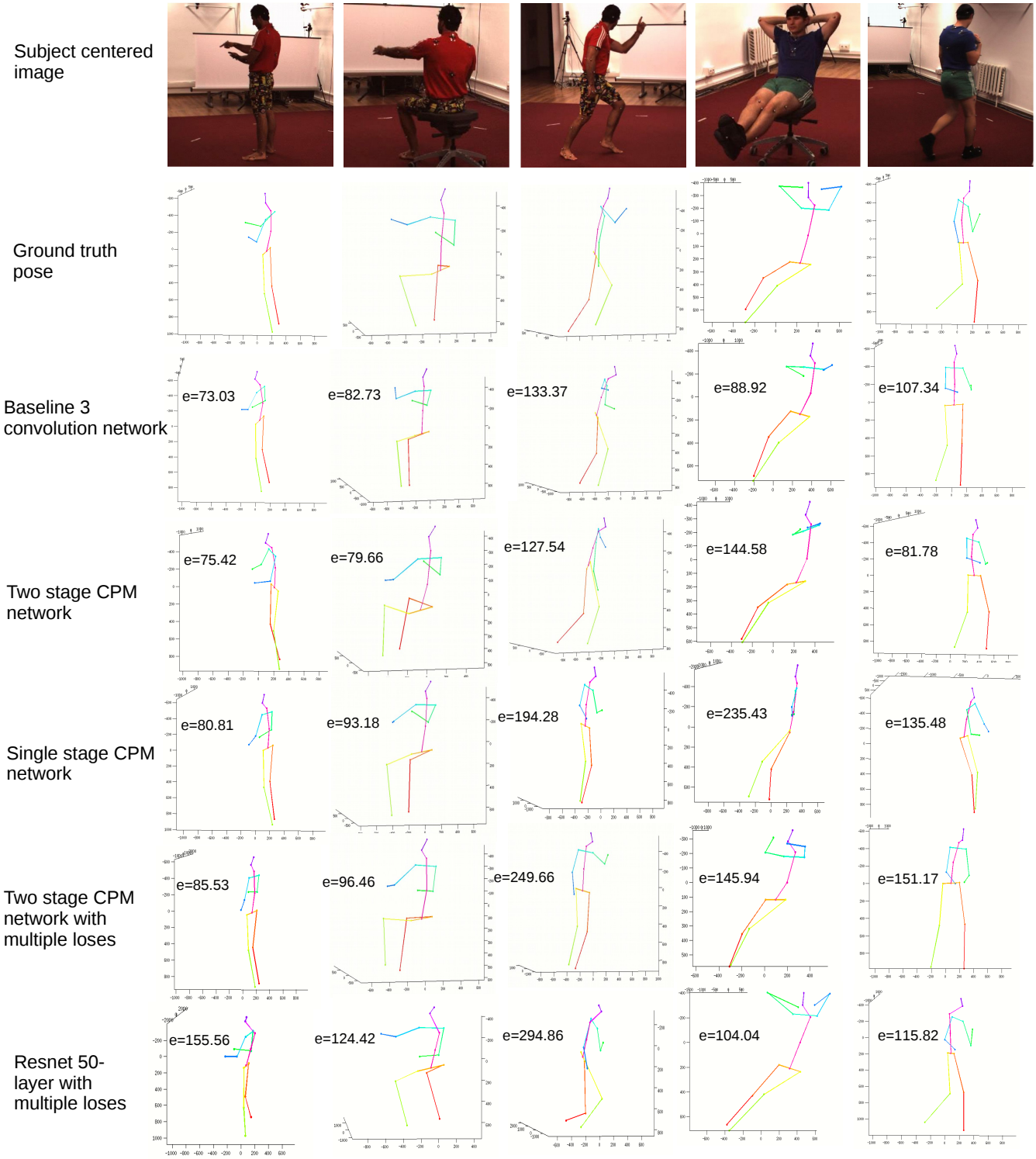

Figure 4.4: Qualitative results for the task of 3D human pose estimation evaluated on the Human3.6M dataset. The figure shows the result of our different architectures with the corresponding error.

Figure 4.5: Results of the 2D pose estimation on the Human3.6M dataset. The five images evaluated above are taken from the S_9 and S_{11} testing actors in the dataset. The figure shows the ground truth poses and then the poses estimated by our two residual architectures. It can be seen that the architecture with only a single euclidean loss for the task of 2D pose estimation gives better results.

5 Conclusion

5.1 Summary

In the thesis we investigate different architectures for the task of 3D human pose estimation. We evaluate our results on the Human3.6M dataset and compare our results with the state of the art. We firstly train a smaller baseline network consisting of three convolutional layers individually for all the 15 activities present in the Human3.6M dataset. It is visible from the results that this baseline network performs very well. This is because each of these networks are specifically trained for a single activity thus they are better adapted to estimate results in that certain pose space and environment. We then train a two stage network based on the state of the art work by [Shih-En Wei & Sheikh 16] for the accurate prediction of 2D parts in monocular images. The first stage of the network purely works as a 2D heat map generator providing predictions of the 2D joints. This information is fused with the second stage of the network which then further extract features for the task of 3D pose estimation. We achieve state of the art results on this architecture. We also train a single stage network without the 2D predictor and see that this architecture indeed under performs the complete two stage architecture. Since we observe some over fitting in our complete two stage CPM architecture, we added another loss for the 2D pose estimation in the network and jointly train the network for both 2D and 3D pose estimation. We see that adding an extra loss layer does not help in our case and this network under performs our two stage CPM network with a single euclidean loss for the task of 3D pose estimation.

We also perform experiments with the residual network after being inspired by the work of [Xingyi Zhou & Wei 16]. We train a 50-layer residual network architecture with multiple losses both for 2D pose estimation and 3D pose estimation and we find that this network under performs the network trained for only 3D pose estimation trained by [Xingyi Zhou & Wei 16]. For completeness, we also train a 50-layer residual network for only the task of 2D pose estimation and find out that this network out performs our 50-layer residual network with multiple losses which confirms to us that residual network performs better when trained for a single task rather than training for multiple tasks with multiple losses.

5.2 Future Work Directions

The human kinematic pose space is very large to be sampled regularly and densely, thus the state of the art datasets for the task of 3D human pose estimation only collect data by focusing on a set of poses which are likely to be of interest because they are common in urban and office environments. This is not very ideal because these datasets only have a small number of actors which does not account for the huge body shape, size and skin color variability in the real world. Moreover, the type of clothing used by the actors is very limited and only focused on a certain part of the world which cannot be easily generalized given the complex multi color clothing used in different parts of the world. Many of these problems have been a bottleneck for task of human 3D pose estimations in the general complex environments. [Rogez & Schmid 16] provides a way of synthesizing a very large number of new in-the-wild images showing more pose configurations and, importantly, providing the corresponding 3D pose annotations. They achieve this synthesizing in a two stage process. The first stage is a motion capture mosaic construction stage that stitches the image patches together. And the second stage is the pose aware blending process that improves the image quality and erases patch seams. Since the images can be manually annotated of 2D poses, the synthetic images for any environment can be generated together with the corresponding 3D poses using the poses from an existing motion capture database.

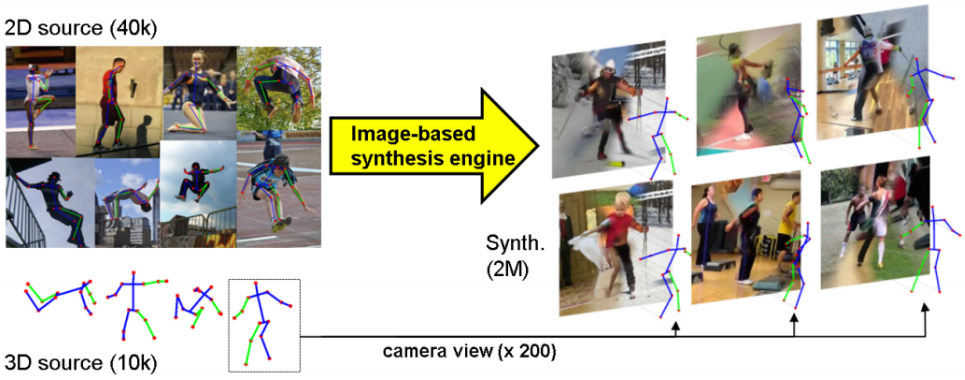

Figure 5.1: Image-based synthesis engine. Input: real images with manual annotation of 2D poses, and 3D poses captured with a Motion Capture (MoCap) system. Taken from [Rogez & Schmid 16].

[Yasin & Iqbal^{+} 16] also devised a dual source approach where one source is the images with annotated 2D poses and the second source is the accurate 3D motion capture data from an existing database. In their work, the 2D poses are estimated from the images and then the k-nearest 3D poses are retrieved from the second

source which are then further refined. This method can be improved if the training is moved to an end to end fashion rather than retrieving the 3D poses on the fly.

For the work of 3D pose estimation to move forward and be able to estimate accurate 3D poses in general complex environments, more work needs to be done in the direction of generating accurate and kinematic-ally plausible synthetic images emulating complex environments and the corresponding accurate 3D annotations. This data can then be used to train end to end regression networks which can give us accurate 3D pose estimations for general environments.

List of Figures

List of Tables

Bibliography

[Alex Krizhevsky & Hinton 12] I. S. Alex Krizhevsky, G. E. Hinton. Imagenet classification with deep convolutional neural networks. 2012.

[Andrew M. Saxe & Ganguli 14] J. L. M. Andrew M. Saxe, S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. 2014.

[Belagiannis & Amin$^+$ 14] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, S. Ilic. 3d pictorial structures for multiple human pose estimation. June 2014.

[Belagiannis & Wang$^+$ 14] V. Belagiannis, X. Wang, B. Schiele, P. Fua, S. Ilic, N. Navab. Multiple human pose estimation with temporally consistent 3d pictorial structures. In *Computer Vision-ECCV 2014 Workshops*, pp. 742–754. Springer International Publishing, 2014.

[Bogo & Kanazawa$^+$ 16] F. Bogo, A. Kanazawa, C. Lassner, P. V. Gehler, J. Romero, M. J. Black. Keep it SMPL: automatic estimation of 3d human pose and shape from a single image. *CoRR*, Vol. abs/1607.08128, 2016.

[Bottou 12] L. Bottou. *Stochastic Gradient Tricks*, Vol. 7700, 430–445. Springer, January 2012.

[Bradley 10] D. Bradley. Learning in modular systems. In *PhD thesis, Robotics Institute, Carnegie Mellon University.*, 2010.

[Bugra Tekin & Fua 16] M. S. Bugra Tekin, Isinsu Katirciogu, P. Fua. Structured prediction of 3d human pose with deep neural networks. 2016.

[Burenius & Sullivan$^+$ 13] M. Burenius, J. Sullivan, S. Carlsson. 3d pictorial structures for multiple view articulated pose estimation. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013.

[Catalin Ionescu & Sminchisescu 14] V. O. Catalin Ionescu, Dragos Papave, C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. In *IEEE Transactions of pattern analysis and machine intelligence*, 2014.

[Cherian & Mairal$^+$ 14] A. Cherian, J. Mairal, K. Alahari, C. Schmid. Mixing body-part sequences for human pose estimation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[Eichner & Marin-Jimenez+ 12] M. Eichner, M. Marin-Jimenez, A. Zisserman, V. Ferrari. 2d articulated human pose estimation and retrieval in (almost) unconstrained still images. *International Journal of Computer Vision*, Vol. 99, pp. 190–214, 2012.

[Felzenszwalb & Huttenlocher 05] P. F. Felzenszwalb, D. P. Huttenlocher. Pictorial structures for object recognition. *Int. J. Comput. Vision*, Vol. 61, No. 1, pp. 55–79, Jan. 2005.

[Glorot & Bengio 10] X. Glorot, Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[Han & Moraga 95] J. Han, C. Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. pp. 195–201, 1995.

[He & Zhang+ 15] K. He, X. Zhang, S. Ren, J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, Vol., 2015.

[Hornik 88] K. Hornik. Multilayer feedforward networks are universal approximators. sep 1988.

[Hubel & Wiesel 68] D. Hubel, T. Wiesel. Receptive fields and functional architecture of monkey striate cortex. 1968.

[Ioffe & Szegedy 15] S. Ioffe, C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[Jia & Shelhamer+ 14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, Vol., 2014.

[Jiang & Martin 08] H. Jiang, D. R. Martin. Global pose estimation using non-tree models. In *CVPR*. IEEE Computer Society, 2008.

[John Duchi & Singer 11] E. H. John Duchi, Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *The Journal of Machine Learning Research*, 2011.

[Kingma & Ba 14] D. P. Kingma, J. Ba. Adam: A method for stochastic optimization. *CoRR*, Vol. abs/1412.6980, 2014.

[Kostrikov & Gall 14] I. Kostrikov, J. Gall. Depth sweep regression forests for estimating 3d human pose from images. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[LeCun & Bottou+ 98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, Vol. 86, pp. 2278–2324, 1998.

[LeCun Yan & Denker 89] B. B. LeCun Yan, J. Denker. Backpropagation applied to handwritten zip code recognition. In *Neural computation 1.4*, 1989.

[LeCun 92] Y. LeCun. A theoretical framework for back-propagation. In P. Mehra, B. Wah, editors, *Artificial Neural Networks: concepts and theory*, Los Alamitos, CA, 1992. IEEE Computer Society Press.

[Lee & Recht$^+$ 10] J. Lee, B. Recht, R. Salakhutdinov, N. Srebro, J. Tropp. Practical large-scale optimization for max-norm regularization. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pp. 1297–1305. 2010.

[Li & Chan 14] S. Li, A. B. Chan. 3d human pose estimation from monocular images with deep convolutional neural network. In *Computer Vision - ACCV 2014 - 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part II*, pp. 332–347, 2014.

[Loper & Mahmood$^+$ 15] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, M. J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, Vol. 34, No. 6, pp. 248:1–248:16, Oct. 2015.

[Marinescu & Dechter 09] R. Marinescu, R. Dechter. And/or branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.*, Vol. 173, No. 16-17, pp. 1457–1491, Nov. 2009.

[Nair & Hinton 10] V. Nair, G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[Pishchulin & Insafutdinov$^+$ 16] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[R. Ghosh & Bagirov 05] J. Y. R. Ghosh, M. Ghosh, A. Bagirov. Comparative analysis of genetic algorithm, simulated annealing and cutting angle method for artificial neural networks. pp. 62–70, 2005.

[Rogez & Schmid 16] G. Rogez, C. Schmid. Mocap-guided data augmentation for 3d pose estimation in the wild. *CoRR*, Vol. abs/1607.02046, 2016.

[Ross Girshick 11] P. K. A. C. A. F. Ross Girshick, Jamie Shotton. Efficient regression of general-activity human poses from depth images. In *ICCV*. IEEE, October 2011.

[Russakovsky & Deng$^+$ 14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, F. Li.

Imagenet large scale visual recognition challenge. *CoRR*, Vol. abs/1409.0575, 2014.

[Shih-En Wei & Sheikh 16] T. K. Shih-En Wei, Varun Ramakrishna, Y. Sheikh. Convolutional pose machines. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[Sigal & Balan+ 10] L. Sigal, A. Balan, M. J. Black. HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International Journal of Computer Vision*, Vol. 87, No. 1, pp. 4–27, March 2010.

[Srivastava & Hinton+ 14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15, pp. 1929–1958, 2014.

[Sungheon Park & Kwak 16] J. H. Sungheon Park, N. Kwak. 3d human pose estimation using convolutional neural networks with 2d pose information. 2016.

[Tekin & Rozantsev+ 15] B. Tekin, A. Rozantsev, V. Lepetit, P. Fua. Direct prediction of 3d body poses from motion compensated sequences. *CoRR*, Vol. abs/1511.06692, 2015.

[Toshev & Szegedy 13] A. Toshev, C. Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, Vol. abs/1312.4659, 2013.

[Varun Ramakrishna & Sheikh 14] M. H. J. A. B. Varun Ramakrishna, Daniel Munoz, Y. Sheikh. Pose machines: Articulated pose estimation via inference machines. sep 2014.

[Xiaowei Zhou & Daniilidis 16] S. L. Xiaowei Zhou, Menglong Zhu, K. Daniilidis. Sparseness meets deepness: 3d human pose estimation from monocular video. 2016.

[Xingyi Zhou & Wei 16] W. Z. S. L. Xingyi Zhou, Xiao Sun, Y. Wei. Deep kinematic pose regression. September 2016.

[Yasin & Iqbal+ 16] H. Yasin, U. Iqbal, B. Krüger, A. Weber, J. Gall. 3d pose estimation from a single monocular image. *CoRR*, Vol. abs/1509.06720, 2016.